**Amazon Delivery Time Prediction System**

**Project Documentation**

**🧩 1. Introduction**

In the modern e-commerce landscape, **timely delivery** plays a crucial role in customer satisfaction. With companies like Amazon handling millions of orders daily, predicting the **estimated delivery time** accurately becomes a key operational challenge.

This project aims to develop a **Machine Learning–based predictive system** that estimates the delivery time for Amazon orders based on various factors such as **weather, traffic, agent performance, and location data**.  
The model is deployed as a user-friendly **Streamlit web application** that allows users to input order details and instantly obtain delivery time predictions.

**🎯 2. Objectives**

* To analyze real-world delivery data and identify factors influencing delivery time.
* To preprocess and clean the dataset for machine learning.
* To perform **Exploratory Data Analysis (EDA)** and visualize key relationships.
* To implement and compare multiple regression models.
* To select the best-performing model (XGBoost) and integrate it with a Streamlit interface.
* To deploy the final app on **Streamlit Cloud** for public access.

**📦 3. Dataset Overview**

The dataset contains details about delivery agents, order times, weather, traffic, and geographical coordinates.

| **Column Name** | **Description** |
| --- | --- |
| Order\_ID | Unique order identifier |
| Agent\_Age | Age of the delivery agent |
| Agent\_Rating | Performance rating of the agent |
| Store\_Latitude / Longitude | Coordinates of the store |
| Drop\_Latitude / Longitude | Coordinates of the delivery location |
| Weather | Weather condition (Sunny, Rainy, Foggy, etc.) |
| Traffic | Traffic density (Low, Medium, High) |
| Vehicle | Type of delivery vehicle |
| Area | Delivery area type (Urban, Rural, Semi-Urban) |
| Category | Order type (Grocery, Food, Electronics, etc.) |
| Delivery\_Time | Target variable (minutes) |
| Distance\_km | Derived feature: distance between store and drop |
| Order\_Year, Month, Day, Hour | Order time details |
| Pickup\_Hour, Pickup\_Minute | Pickup timing |
| Pickup\_Delay\_Minutes | Delay between order and pickup |

**🧹 4. Data Preprocessing**

The raw dataset contained missing values, inconsistent formats, and redundant columns. The following preprocessing steps were performed:

1. **Handling Missing Values**
   * Imputed missing categorical values with mode.
   * Imputed missing numerical values with median.
2. **Removing Duplicates**
   * Dropped duplicate rows to ensure data consistency.
3. **Feature Encoding**
   * Converted categorical columns (Weather, Traffic, Vehicle, Area, Category) using **One-Hot Encoding**.
   * Normalized continuous features where required.
4. **Outlier Treatment**
   * Detected outliers using **IQR (Interquartile Range)** method.
   * Replaced extreme outliers to avoid skewed model behavior.
5. **Feature Scaling**
   * Applied **StandardScaler** to normalize numerical columns for models sensitive to scale.
6. **Train-Test Split**
   * Divided dataset into **80% training** and **20% testing** data.

**🔬 5. Exploratory Data Analysis (EDA)**

EDA was performed to gain insights into patterns affecting delivery times.

**Key Observations:**

* **Traffic & Weather** have a major impact — High traffic and rainy weather lead to higher delivery times.
* **Agent Rating** negatively correlates with delay — more experienced agents tend to deliver faster.
* **Distance\_km** is directly proportional to delivery time.
* **Urban areas** show shorter delivery times due to proximity of hubs.

**Visualizations:**

* Distribution plots for Delivery Time.
* Heatmap showing feature correlations.
* Bar charts for average delivery time vs. traffic/weather.
* Scatter plots between distance and delivery time.

**⚙️ 6. Feature Engineering**

New derived features were created to improve model accuracy:

| **Feature Name** | **Description** |
| --- | --- |
| Distance\_km | Calculated using store and drop coordinates (Haversine formula) |
| Pickup\_Delay\_Minutes | Difference between pickup and order time |
| Order\_DayOfWeek | Derived from date (0=Mon, 6=Sun) |
| Order\_Hour | Extracted from timestamp |
| Time\_of\_Day | Categorical grouping (Morning/Afternoon/Evening/Night) |

These features enhanced the temporal and spatial understanding of deliveries.

**🤖 7. Model Development**

Several regression models were trained and evaluated:

| **Model** | **R² Score** | **RMSE (minutes)** |
| --- | --- | --- |
| Linear Regression | 0.62 | 12.8 |
| Random Forest Regressor | 0.79 | 8.6 |
| Gradient Boosting Regressor | 0.82 | 7.9 |
| **XGBoost Regressor (Final)** | **0.86** | **7.4** |

**Why XGBoost?**

* Handles non-linear relationships effectively.
* Robust to outliers.
* Excellent generalization performance.
* Lower RMSE and faster inference time.

**🧪 8. Model Evaluation**

The XGBoost model was evaluated using:

* **Mean Absolute Error (MAE):** Measures average absolute deviation.
* **Root Mean Squared Error (RMSE):** Penalizes large errors.
* **R² Score:** Explains the proportion of variance captured.

**Results:**

* R² Score: **0.86**
* RMSE: **7.4 minutes**
* MAE: **5.8 minutes**

This means predictions were generally within ±10 minutes of actual delivery time.

**🧩 9. Model Serialization**

To deploy the model efficiently:

* The trained model pipeline (preprocessing + XGBoost) was saved as
* joblib.dump(pipeline, "xgboost\_pipeline.pkl")
* Loaded back in the Streamlit app using:
* loaded\_model = joblib.load("xgboost\_pipeline.pkl")

This ensured consistent preprocessing between training and deployment.

**💻 10. Streamlit Application**

**App Features:**

* Top navigation with pages:
  + 🏠 **Home** — Project intro
  + 🔮 **Prediction** — Interactive form to predict delivery time
  + 📊 **Insights** — Basic charts for traffic and weather trends
  + 📬 **Contact** — Links to developer profile

**User Input Fields:**

* Agent, order, and location details.
* Dropdowns for weather, traffic, vehicle, and area.
* Numeric inputs for coordinates and time.

**Prediction Output:**

After user input, clicking **“Predict Delivery Time”** displays:

⏱ Predicted Delivery Time: 17.21 minutes

**Insights Page:**

* Visualizes average delivery time by traffic and weather.
* Includes histogram of delivery time distribution.

**☁️ 11. Deployment**

The application was deployed using **Streamlit Cloud**.

**Deployment Steps:**

1. Uploaded the project repository to GitHub.
2. Linked it to **Streamlit Cloud Dashboard**.
3. Set the main file path to app.py.
4. Added required dependencies in requirements.txt.
5. Deployed successfully at:  
   🔗 [**https://amazon-delivery-time-prediction-system.streamlit.app/**](https://amazon-delivery-time-prediction-system.streamlit.app/)

**📈 12. Results and Insights**

* The system accurately predicts delivery times for different conditions.
* Traffic and weather were the most influential features.
* XGBoost provided the best performance with minimal overfitting.
* The interactive Streamlit UI makes it accessible to non-technical users.

**🔮 13. Future Enhancements**

* Integrate **real-time traffic and weather APIs** for live prediction.
* Add **map visualization** using geolocation APIs.
* Include **deep learning (LSTM)** for time-dependent prediction.
* Expand dataset with delivery partner data and timestamps.

**👩‍💻 14. Conclusion**

This project successfully developed a full-cycle **machine learning pipeline** that predicts Amazon delivery times using real-world parameters.  
From preprocessing and modeling to web deployment, the system demonstrates the power of data-driven logistics optimization.

The app is now publicly accessible, offering fast and accurate predictions, paving the way for **smarter, AI-driven delivery management**.

**📚 15. References**

* Scikit-learn Documentation — <https://scikit-learn.org/>
* XGBoost Documentation — <https://xgboost.readthedocs.io/>
* Streamlit Documentation — <https://docs.streamlit.io/>
* Pandas, NumPy, Matplotlib official docs

**GUNTUR RIDHI**

**gunturridhi@gmail.com**