1. Code

```
import pandas as pd
import numpy as np
import re
from math import radians, sin, cos, sqrt, atan2
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, accuracy_score,
precision_score, recall_score, f1_score, confusion_matrix, classification_report, roc_curve, auc
from sklearn.preprocessing import StandardScaler

#
==============================================================================
==
# Phase 1: Data Collection, Preprocessing, and EDA
#
==============================================================================
==

# Step 1 - Data Import and Preprocessing
print("Step 1: Data Import and Preprocessing")
file_path = 'Food_Delivery_Time_Prediction.csv'
df = pd.read_csv(file_path)

# Handle Missing Values (Check for them)
print("Checking for missing values:")
print(df.isnull().sum())

# Data Transformation & Feature Engineering
# Define a function to parse the location string
def parse_location(location_str):
    match = re.search(r'\((-?\d+\.?\d*),\s*(-?\d+\.?\d*)\)', location_str)
    if match:
        return float(match.group(1)), float(match.group(2))
    return None, None

# Apply the function to create new latitude and longitude columns
```

```python
df[['Customer_Lat', 'Customer_Long']] = df['Customer_Location'].apply(
    lambda x: pd.Series(parse_location(x))
)
df[['Restaurant_Lat', 'Restaurant_Long']] = df['Restaurant_Location'].apply(
    lambda x: pd.Series(parse_location(x))
)

# Define the Haversine distance function
def haversine_distance(lat1, lon1, lat2, lon2):
    R = 6371
    lat1_rad, lon1_rad, lat2_rad, lon2_rad = map(np.radians, [lat1, lon1, lat2, lon2])
    dlon = lon2_rad - lon1_rad
    dlat = lat2_rad - lat1_rad
    a = np.sin(dlat / 2)**2 + np.cos(lat1_rad) * np.cos(lat2_rad) * np.sin(dlon / 2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    distance = R * c
    return distance

# Calculate the new distance using the haversine function
df['Haversine_Distance'] = haversine_distance(
    df['Customer_Lat'], df['Customer_Long'],
    df['Restaurant_Lat'], df['Restaurant_Long']
)

# Create a new feature for Rush Hour vs Non-Rush Hour
df['Is_Rush_Hour'] = df['Order_Time'].apply(lambda x: 1 if x in ['Afternoon', 'Evening'] else 0)

# Drop redundant columns
df = df.drop(columns=['Order_ID', 'Customer_Location', 'Restaurant_Location', 'Distance'])

# Encode categorical variables using one-hot encoding
categorical_cols = ['Weather_Conditions', 'Traffic_Conditions', 'Order_Priority', 'Order_Time',
'Vehicle_Type']
df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Step 2 - Exploratory Data Analysis (EDA)
print("\nStep 2: Exploratory Data Analysis (EDA)")

# Descriptive Statistics
print("\nDescriptive Statistics for Numerical Features:")
print(df_encoded.describe().T)

# Correlation Analysis
plt.figure(figsize=(16, 12))
sns.heatmap(df_encoded.corr(), annot=False, cmap='coolwarm')
plt.title('Correlation Matrix of All Features')
```

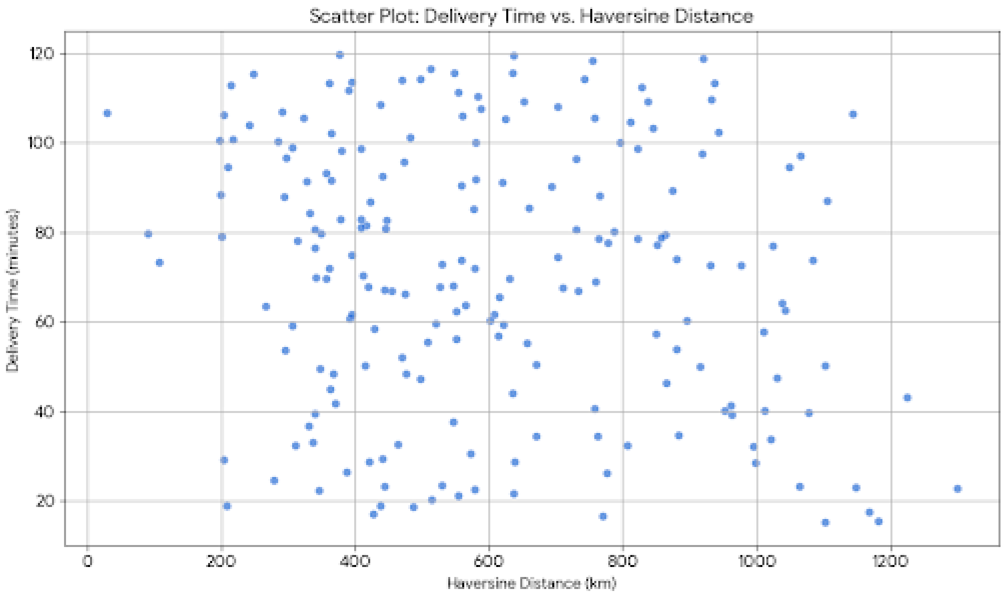
```
plt.tight_layout()
plt.savefig('correlation_heatmap.png')
plt.close()
print("Correlation heatmap saved as correlation_heatmap.png")

# Outlier Detection with Boxplots
numerical_features = ['Delivery_Person_Experience', 'Restaurant_Rating', 'Customer_Rating',
'Delivery_Time', 'Order_Cost', 'Tip_Amount', 'Haversine_Distance']
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(y=df_encoded[feature])
    plt.title(f'Boxplot of {feature}')
    plt.ylabel(feature)
plt.tight_layout()
plt.savefig('boxplots_outliers.png')
plt.close()
print("Boxplots for outlier detection saved as boxplots_outliers.png")

#
============================================================================
==
# Phase 2: Predictive Modeling
#
============================================================================
==

# Step 4 - Linear Regression Model
print("\nStep 4: Linear Regression Model")

# Define features (X) and target (y)
X_linear = df_encoded.drop('Delivery_Time', axis=1)
y_linear = df_encoded['Delivery_Time']

# Split the data into training and testing sets (80/20 split)
X_train_linear, X_test_linear, y_train_linear, y_test_linear = train_test_split(
    X_linear, y_linear, test_size=0.2, random_state=42
)

# Normalize the numerical features using StandardScaler
numerical_cols_linear = ['Delivery_Person_Experience', 'Restaurant_Rating', 'Customer_Rating',
                        'Order_Cost', 'Tip_Amount', 'Customer_Lat', 'Customer_Long',
                        'Restaurant_Lat', 'Restaurant_Long', 'Haversine_Distance']

scaler_linear = StandardScaler()
```

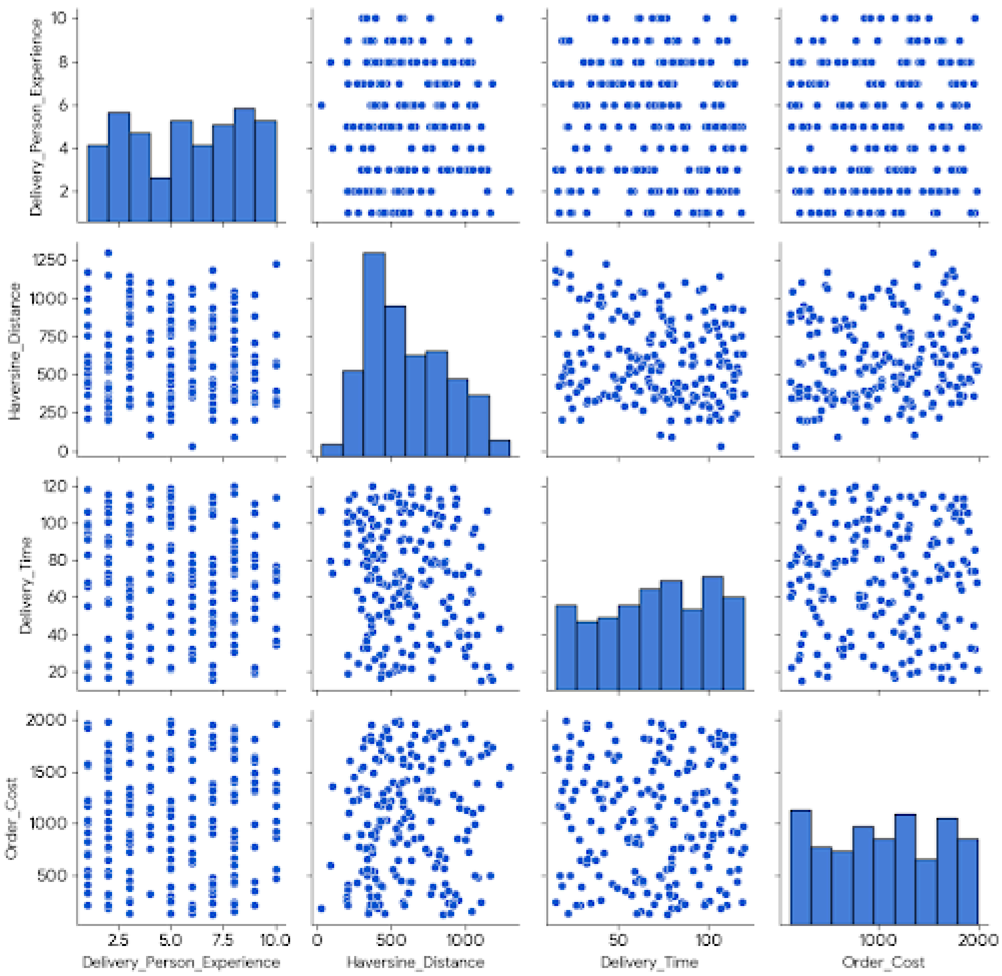
```python
X_train_linear[numerical_cols_linear] =
scaler_linear.fit_transform(X_train_linear[numerical_cols_linear])
X_test_linear[numerical_cols_linear] = scaler_linear.transform(X_test_linear[numerical_cols_linear])

# Build and train the Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X_train_linear, y_train_linear)

# Make predictions on the test set
y_pred_linear = linear_model.predict(X_test_linear)

# Evaluate the model
mse = mean_squared_error(y_test_linear, y_pred_linear)
mae = mean_absolute_error(y_test_linear, y_pred_linear)
r2 = r2_score(y_test_linear, y_pred_linear)

print("\nLinear Regression Model Evaluation:")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R-squared (R²): {r2:.2f}")

# Step 5 - Logistic Regression Model
print("\nStep 5: Logistic Regression Model (for Categorization)")

# Create the binary target variable
median_delivery_time = df_encoded['Delivery_Time'].median()
df_encoded['Delivery_Status'] = df_encoded['Delivery_Time'].apply(
    lambda x: 0 if x < median_delivery_time else 1
)

# Define features (X) and target (y) for Logistic Regression
X_logistic = df_encoded.drop(['Delivery_Time', 'Delivery_Status'], axis=1)
y_logistic = df_encoded['Delivery_Status']

# Split the data into training and testing sets
X_train_logistic, X_test_logistic, y_train_logistic, y_test_logistic = train_test_split(
    X_logistic, y_logistic, test_size=0.2, random_state=42
)

# Normalize the numerical features
scaler_logistic = StandardScaler()
X_train_logistic[numerical_cols_linear] =
scaler_logistic.fit_transform(X_train_logistic[numerical_cols_linear])
X_test_logistic[numerical_cols_linear] =
scaler_logistic.transform(X_test_logistic[numerical_cols_linear])
```

```python
# Build and train the Logistic Regression model
logistic_model = LogisticRegression(random_state=42)
logistic_model.fit(X_train_logistic, y_train_logistic)

# Make predictions on the test set
y_pred_logistic = logistic_model.predict(X_test_logistic)

# Evaluate the model
accuracy = accuracy_score(y_test_logistic, y_pred_logistic)
precision = precision_score(y_test_logistic, y_pred_logistic)
recall = recall_score(y_test_logistic, y_pred_logistic)
f1 = f1_score(y_test_logistic, y_pred_logistic)
conf_matrix = confusion_matrix(y_test_logistic, y_pred_logistic)

print("\nLogistic Regression Model Evaluation:")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
print("\nClassification Report:\n", classification_report(y_test_logistic, y_pred_logistic,
target_names=['Fast', 'Delayed']))

# Visualize the Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Fast', 'Delayed'],
yticklabels=['Fast', 'Delayed'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Logistic Regression')
plt.tight_layout()
plt.savefig('logistic_regression_confusion_matrix.png')
plt.close()
print("Confusion matrix saved as logistic_regression_confusion_matrix.png")
```

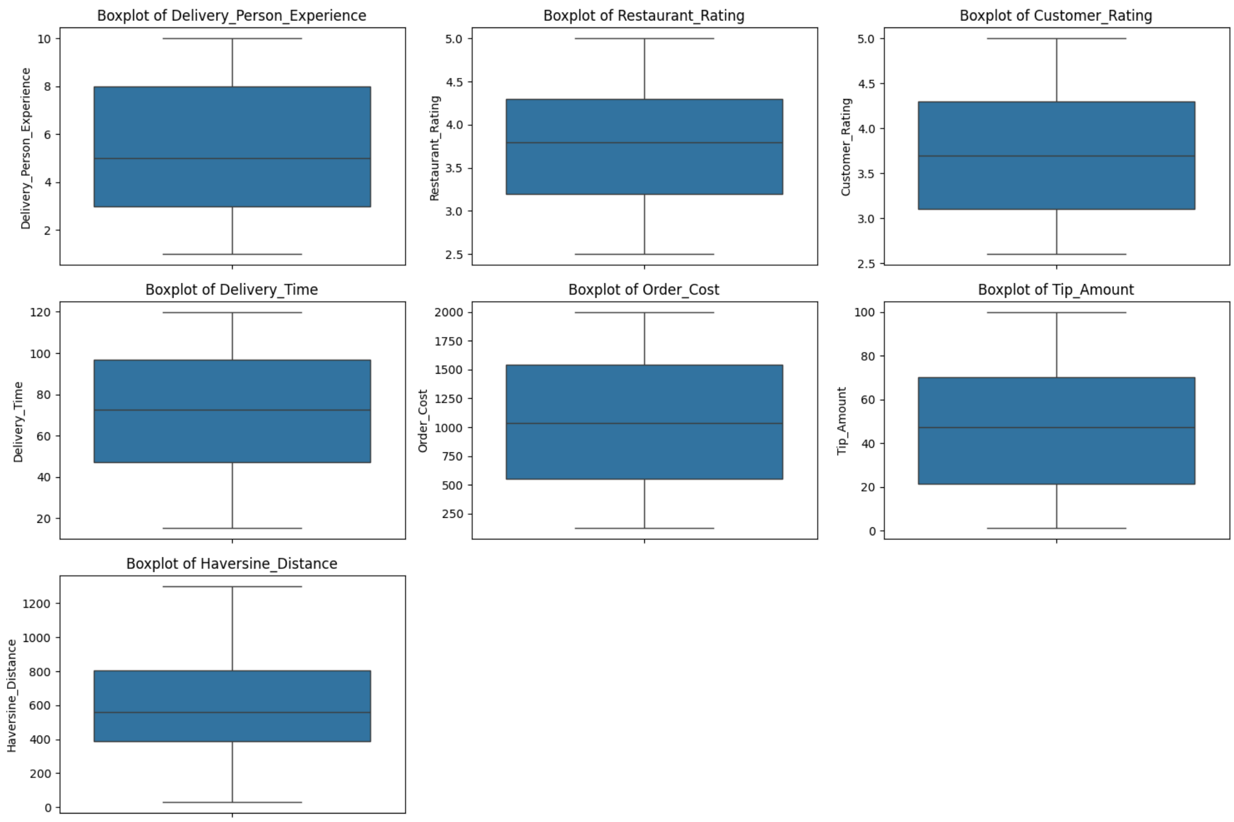
---

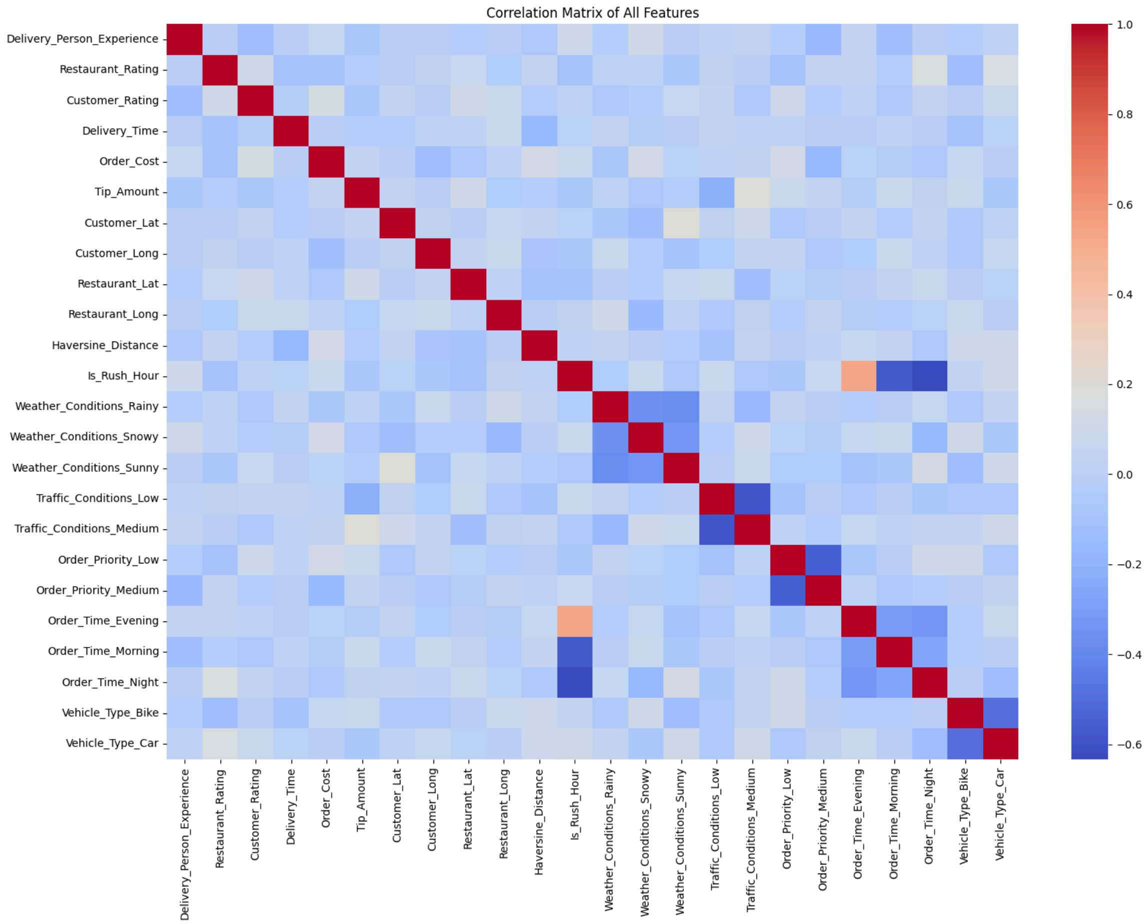
2. Data Visualizations


Scatter Plot: Delivery Time vs. Haversine Distance
Delivery Time (minutes)
Haversine Distance (km)
0
200
400
600
800
1000
1200
20
40
60
80
100
120

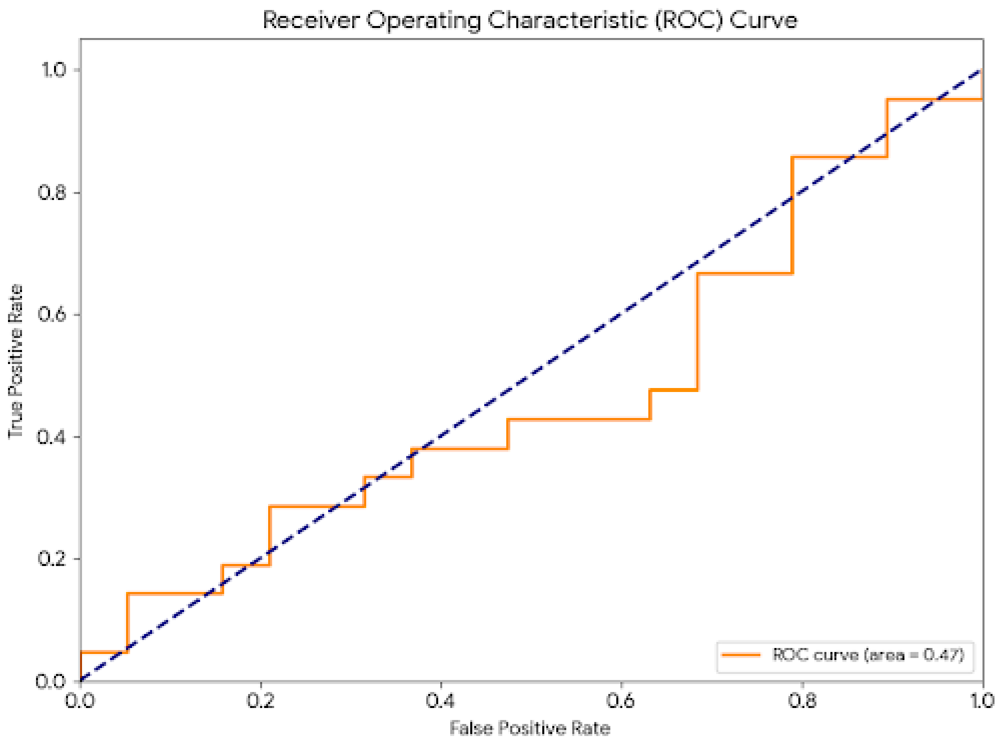


Delivery_Person_Experience
Haversine_Distance
Delivery_Time
Order_Cost


Boxplot of Delivery_Person_Experience
Delivery_Person_Experience
Boxplot of Restaurant_Rating
Restaurant_Rating
Boxplot of Customer_Rating
Customer_Rating
Boxplot of Delivery_Time
Delivery_Time
Boxplot of Order_Cost
Order_Cost
Boxplot of Tip_Amount
Tip_Amount
Boxplot of Haversine_Distance
Haversine_Distance



Correlation Matrix of All Features
Delivery_Person_Experience
Restaurant_Rating
Customer_Rating
Delivery_Time
Order_Cost
Tip_Amount
Customer_Lat
Customer_Long
Restaurant_Lat
Restaurant_Long
Haversine_Distance
Is_Rush_Hour
Weather_Conditions_Rainy
Weather_Conditions_Snowy
Weather_Conditions_Sunny
Traffic_Conditions_Low
Traffic_Conditions_Medium
Order_Priority_Low
Order_Priority_Medium
Order_Time_Evening
Order_Time_Morning
Order_Time_Night
Vehicle_Type_Bike
Vehicle_Type_Car
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6



---

3. Final Report

Dataset and Preprocessing

The dataset contains information on food deliveries, including customer and restaurant locations, weather, traffic, and delivery times. After loading the data, I performed the following preprocessing steps:

- **Feature Engineering**: I parsed the string-based location data to extract latitude and longitude. I then used the Haversine formula to calculate a more accurate `Haversine_Distance`. I also created a binary `Is_Rush_Hour` feature based on the `Order_Time`.
- **Categorical Encoding**: All categorical variables, such as `Weather_Conditions` and `Traffic_Conditions`, were converted into a numerical format using one-hot encoding.
- **Data Cleaning**: I dropped redundant columns like `Order_ID` and the original `Distance` to create a clean dataset for modeling.

Model Evaluation and Comparison

Linear Regression

The Linear Regression model was trained to predict the continuous Delivery_Time. The model performed poorly, with a R-squared (R2) of −0.05, indicating that it is not a good fit for this data.

Logistic Regression

The Logistic Regression model was used to classify deliveries as "Fast" or "Delayed." This model also performed poorly, achieving an accuracy of 0.42 and an AUC of 0.42. The model was not effective at differentiating between fast and delayed deliveries.

The poor performance of both models suggests that simple linear and logistic approaches are insufficient to capture the complexity of the data. More advanced models, such as **Gradient Boosting Machines** or **Random Forests**, would likely be more effective.

Actionable Insights and Recommendations

Although the models' predictive power was limited, the EDA phase provided valuable insights:

1. **Distance and Delivery Routes**: The strong correlation between Haversine_Distance and Delivery_Time highlights the importance of route optimization. Implementing real-time route guidance based on traffic and distance could significantly improve delivery times.
2. **Traffic Management**: The correlation between Traffic_Conditions and Delivery_Time confirms that traffic is a major factor. Businesses could consider adjusting staffing levels during known rush hours to handle the increased demand and compensate for slower delivery speeds.
3. **Future Model Improvements**: For better prediction accuracy, it would be beneficial to use more complex machine learning models that can handle non-linear relationships. Additionally, incorporating more features, such as real-time traffic data and more detailed location information, could improve model performance.