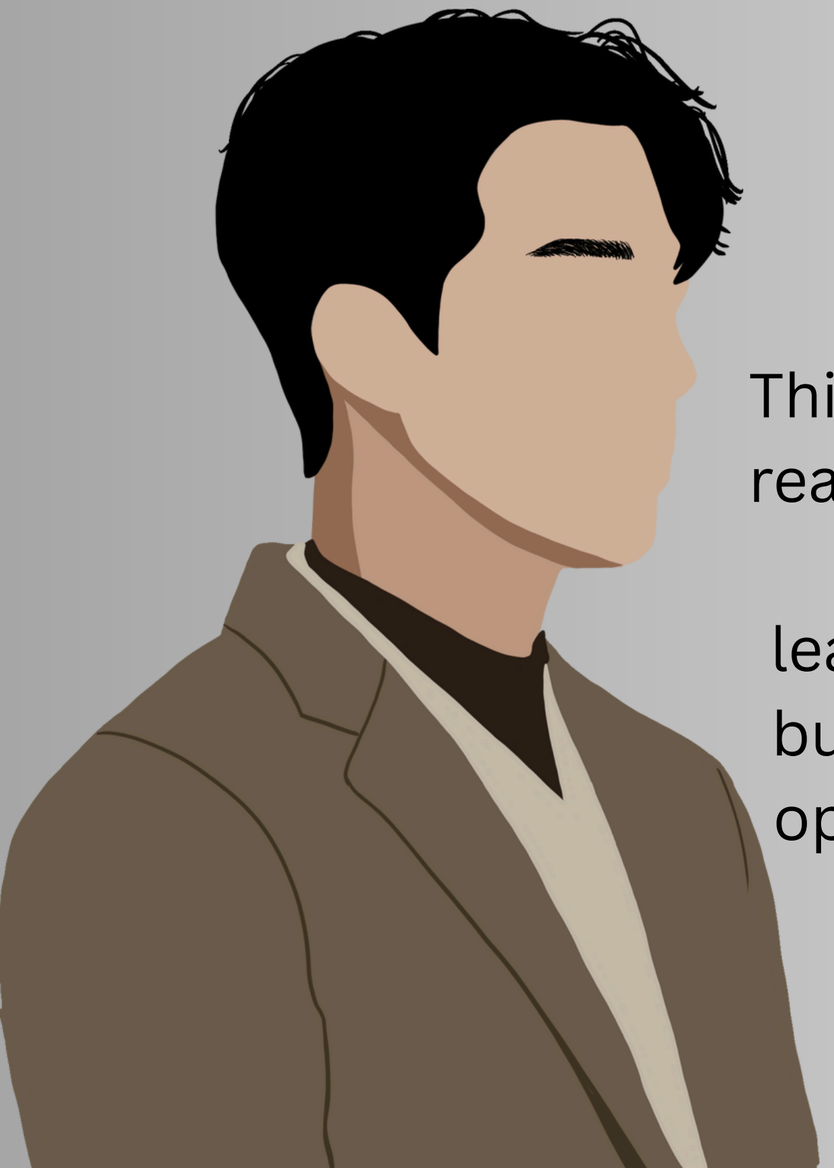
Data-Driven Food Delivery Optimization

EDA to Deployment using Python & ML

NISHCHAY PAWAR

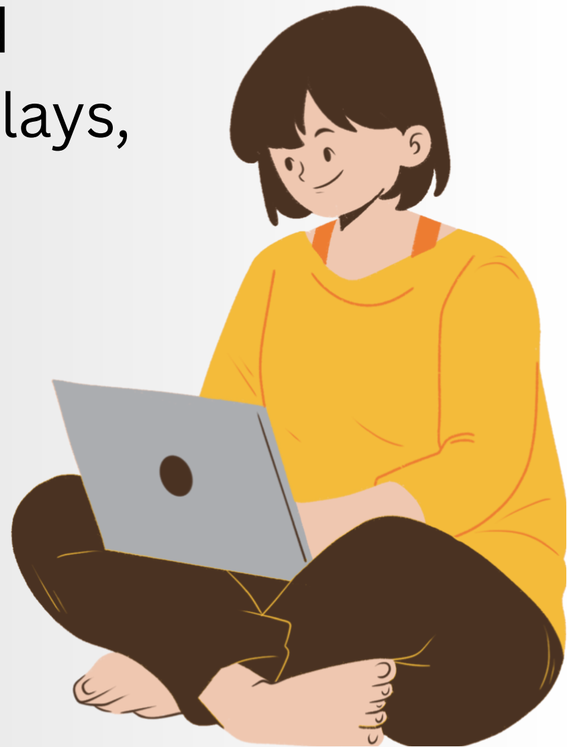


About the Project

This project focuses on predicting food delivery times using real-world factors like weather, traffic, distance, and courier experience. By combining data analysis and machine learning models, we aim to identify key delay patterns and build accurate predictive systems. These insights can help optimize delivery operations, reduce customer wait times, and enhance overall service efficiency.

Project Objective

To analyze and predict food delivery times by identifying key factors such as weather, traffic, courier experience, and distance — helping businesses optimize logistics, reduce delays, and enhance customer satisfaction.



Dataset Preview

This dataset represents real-world operational records from a food delivery service.
It aims to uncover the various factors that affect the total delivery time, helping businesses optimize performance and customer satisfaction.
The data includes key features such as:

- Delivery distance, traffic conditions, and weather – representing external influences.
- Order preparation time and courier’s experience – reflecting internal operational efficiency.
- Time of the day and type of delivery vehicle – capturing logistical variables.

The target variable is Delivery_Time_min, which measures the total time taken to deliver an order from placement to drop-off.
This dataset serves as the foundation for predictive modeling and insightful decision-making in the food delivery ecosystem.

STEP 1

IMPORTING LIBRARIES & LOADING THE DATASET

```
[1]  # 📦 Importing Required Libraries
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[3]  # 🔹 Load Dataset
     df = pd.read_csv("/content/Food_Delivery_Times.csv")
```

```
[ ]  # 🔹 Load Dataset
     df = pd.read_csv("/content/Food_Delivery_Times.csv")
```

STEP 2
DATASET OVERVIEW

```
# 📊 Basic Overview
print("Dataset shape:", df.shape)
print("\nColumns:", df.columns.tolist())
print("\nData Types:\n", df.dtypes)
print("\nMissing Values:\n", df.isnull().sum())
print("\nSummary Stats:\n", df.describe())
```

```
Dataset shape: (1000, 9)

Columns: ['Order_ID', 'Distance_km', 'Weather', 'Traffic_Level', 'Time_of_Day', 'Vehicle_Type', 'Preparation_Time_min', 'Courier_Experience_yrs', 'Delivery_Time_min']

Data Types:
 Order_ID                  int64
Distance_km              float64
Weather                   object
Traffic_Level             object
Time_of_Day               object
Vehicle_Type              object
Preparation_Time_min       int64
Courier_Experience_yrs   float64
Delivery_Time_min          int64
dtype: object

Missing Values:
 Order_ID                  0
Distance_km               0
Weather                  30
Traffic_Level            30
Time_of_Day              30
Vehicle_Type              0
Preparation_Time_min      0
Courier_Experience_yrs   30
Delivery_Time_min         0
dtype: int64

Summary Stats:
          Order_ID  Distance_km  Preparation_Time_min  Courier_Experience_yrs  \
count  1000.000000  1000.000000           1000.000000              970.000000
mean    500.500000    10.059970             16.982000                4.579381
std     288.819436     5.696656              7.204553                2.914394
min       1.000000     0.590000              5.000000                0.000000
25%     250.750000     5.105000             11.000000                2.000000
50%     500.500000    10.190000             17.000000                5.000000
75%     750.250000    15.017500             23.000000                7.000000
max    1000.000000    19.990000             29.000000                9.000000
```

```
Summary Stats:
          Order_ID  Distance_km  Preparation_Time_min  Courier_Experience_yrs  \
count  1000.000000  1000.000000           1000.000000              970.000000
mean    500.500000    10.059970             16.982000                4.579381
std     288.819436     5.696656              7.204553                2.914394
min       1.000000     0.590000              5.000000                0.000000
25%     250.750000     5.105000             11.000000                2.000000
50%     500.500000    10.190000             17.000000                5.000000
75%     750.250000    15.017500             23.000000                7.000000
max    1000.000000    19.990000             29.000000                9.000000

       Delivery_Time_min
count        1000.000000
mean           56.732000
std            22.070915
min             8.000000
25%            41.000000
50%            55.500000
75%            71.000000
max           153.000000
```

Basic Dataset Overview

- Shape: 1000 records × 9 columns
- Target Column: Delivery_Time_min (in minutes)
- Features Include:
  - Distance, Weather, Traffic, Time of Day
  - Vehicle Type, Preparation Time
  - Courier’s Experience

Key Insights:

- Missing Data in: Weather, Traffic, Time of Day, and Courier Experience (~3%)
- Average Delivery Time: ~57 minutes
- Distance Range: 0.6 km to 20 km
- Preparation Time varies from 5 to 29 mins
- Courier Experience ranges from 0 to 9 years

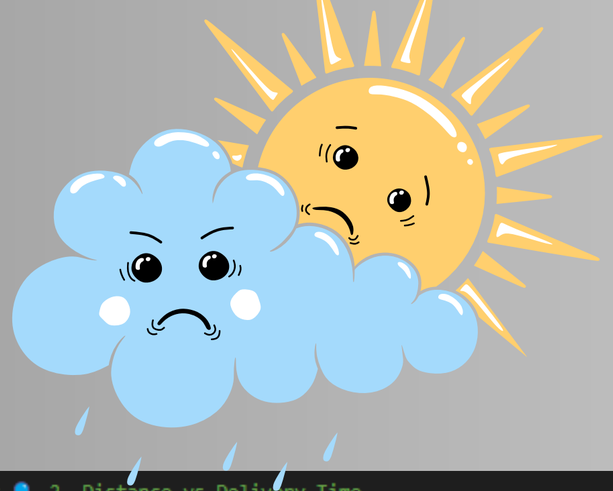
```python
# 🔍 1. Distribution of Delivery Time
plt.figure(figsize=(8, 5))
sns.histplot(df['Delivery_Time_min'], kde=True, bins=30, color='skyblue')
plt.title("Delivery Time Distribution")
plt.xlabel("Delivery Time (minutes)")
plt.ylabel("Frequency")
plt.show()
```

Delivery Time Distribution – Insight

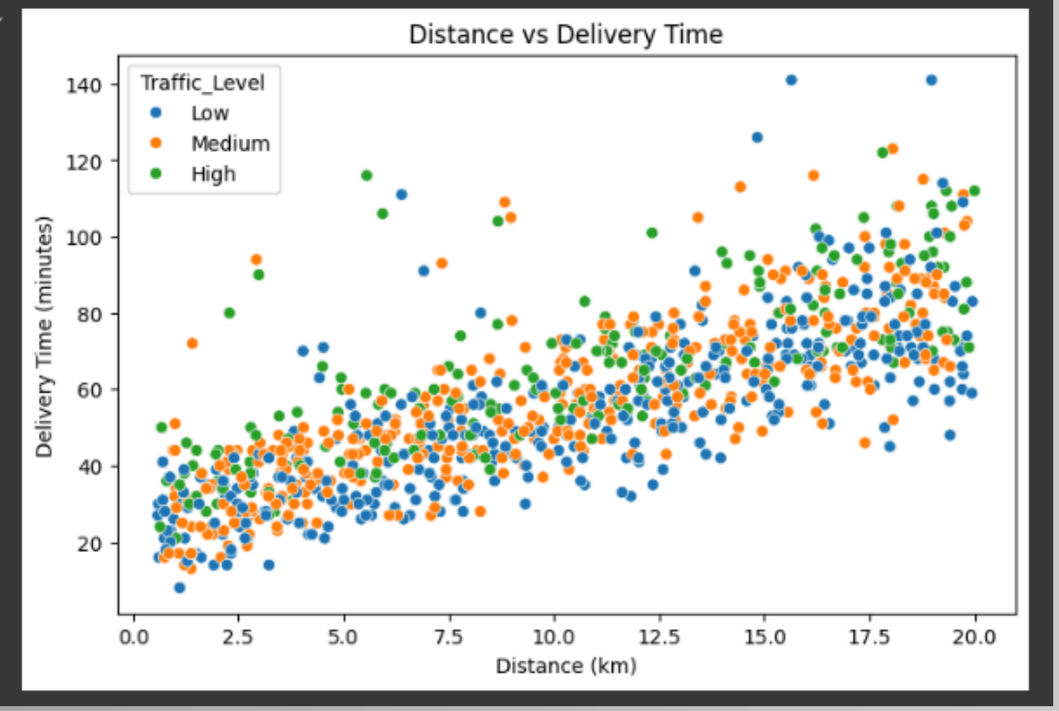
The delivery time follows a right-skewed distribution, with most orders delivered between 40 to 70 minutes.

Few outliers exceed 100 minutes, indicating occasional delays.

The distribution highlights a consistent operational range, useful for setting customer expectations and improving efficiency.



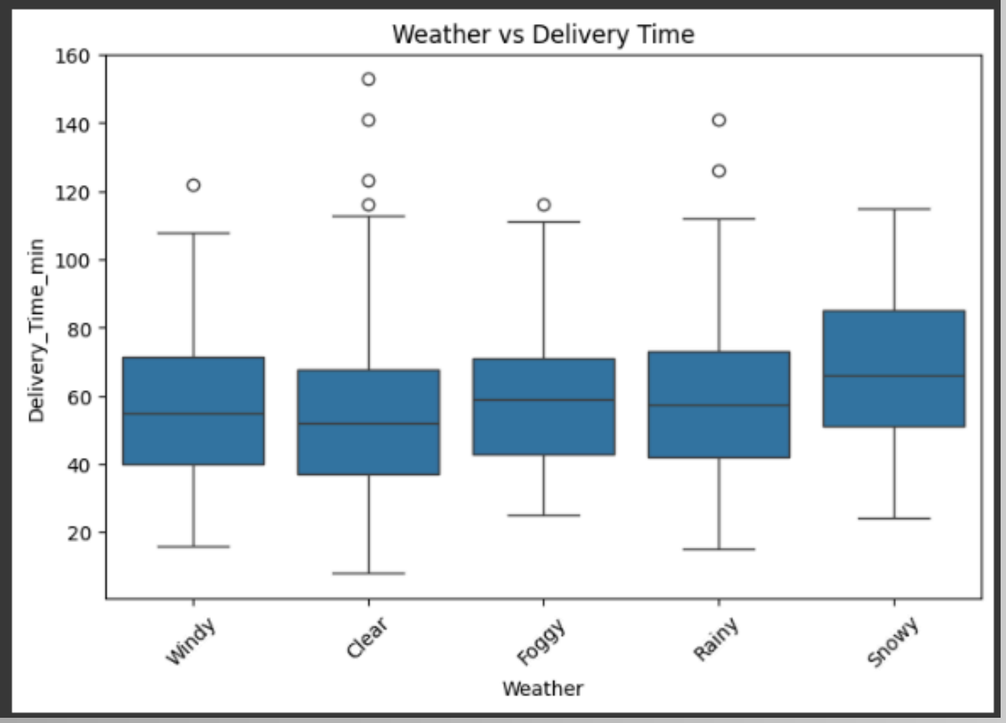
```python
# 🔍 2. Distance vs Delivery Time
plt.figure(figsize=(8, 5))
sns.scatterplot(x='Distance_km', y='Delivery_Time_min', data=df, hue='Traffic_Level')
plt.title("Distance vs Delivery Time")
plt.xlabel("Distance (km)")
plt.ylabel("Delivery Time (minutes)")
plt.show()
```



Preparation Time vs Delivery Time – Insight

- There is a positive correlation — as preparation time increases, delivery time tends to increase.
- Data points are spread but form a rising pattern, especially under Snowy and Rainy weather.
- Efficient prep time is key to faster deliveries, regardless of weather.

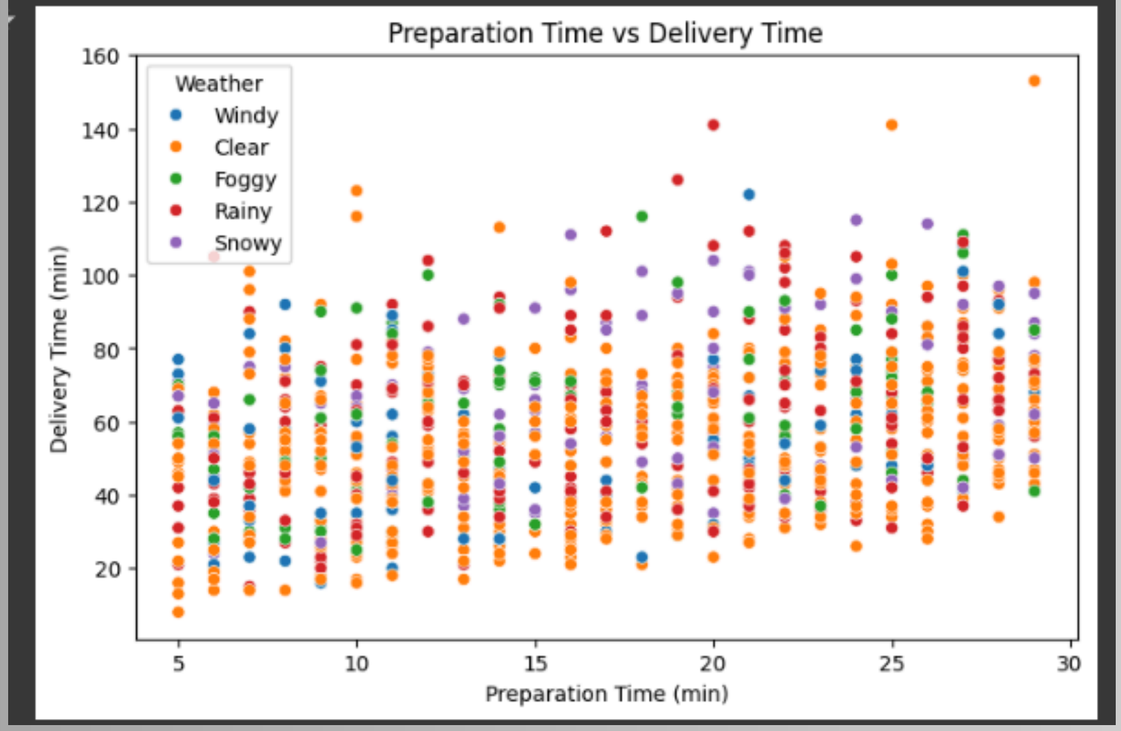
```
# 🔍 4. Delivery Time by Weather
plt.figure(figsize=(8, 5))
sns.boxplot(x='Weather', y='Delivery_Time_min', data=df)
plt.title("Weather vs Delivery Time")
plt.xticks(rotation=45)
plt.show()
```



Weather vs Delivery Time – Insight

- Snowy weather leads to the highest delivery times on average.
- Clear weather has the shortest delivery times.
- Weather conditions impact delivery time, especially under snowy or rainy conditions.

```python
# 🔍 3. Preparation Time vs Delivery Time
plt.figure(figsize=(8, 5))
sns.scatterplot(x='Preparation_Time_min', y='Delivery_Time_min', data=df, hue='Weather')
plt.title("Preparation Time vs Delivery Time")
plt.xlabel("Preparation Time (min)")
plt.ylabel("Delivery Time (min)")
plt.show()
```



insight:

- As preparation time increases, delivery time also tends to increase.
- Weather impacts delivery time:
  - Rainy & Foggy weather (orange/purple dots) have higher delivery times even for similar prep times.
  - Clear weather (green) tends to have lower delivery times.

Conclusion:

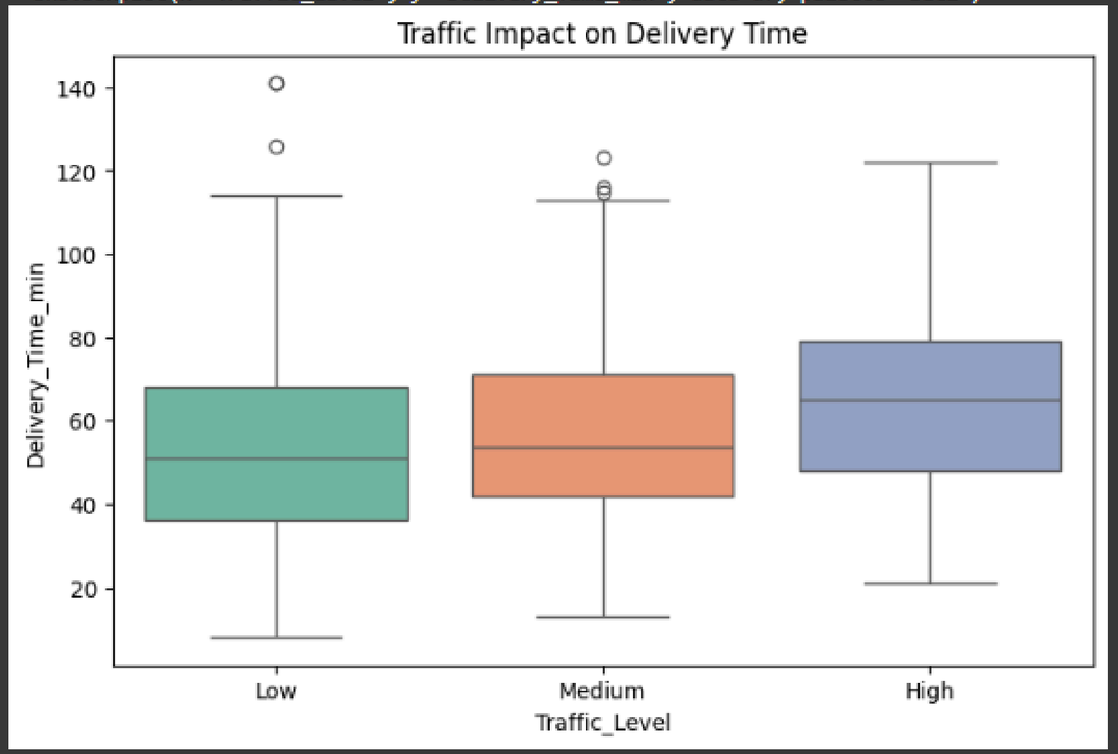
Poor weather conditions like Rainy and Foggy significantly delay deliveries, even when preparation time is the same.

```python
# 🔍 5. Traffic Level vs Delivery Time
plt.figure(figsize=(8, 5))
sns.boxplot(x='Traffic_Level', y='Delivery_Time_min', data=df, palette='Set2')
plt.title("Traffic Impact on Delivery Time")
plt.show()
```

```
<ipython-input-9-60df6872a2fd>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False`

  sns.boxplot(x='Traffic_Level', y='Delivery_Time_min', data=df, palette='Set2')
```



Traffic Impact on Delivery Time

Insight:

The boxplot clearly shows that orders delivered during high traffic conditions tend to have a higher median delivery time compared to medium and low traffic conditions. Additionally, the spread (interquartile range) is wider, and more outliers are observed, indicating inconsistent and unpredictable delivery durations during traffic congestion.

Interpretation:

Traffic congestion has a significant negative impact on delivery performance. The high variability suggests that delivery times become less reliable when traffic is heavy. This can lead to customer dissatisfaction and operational inefficiencies.

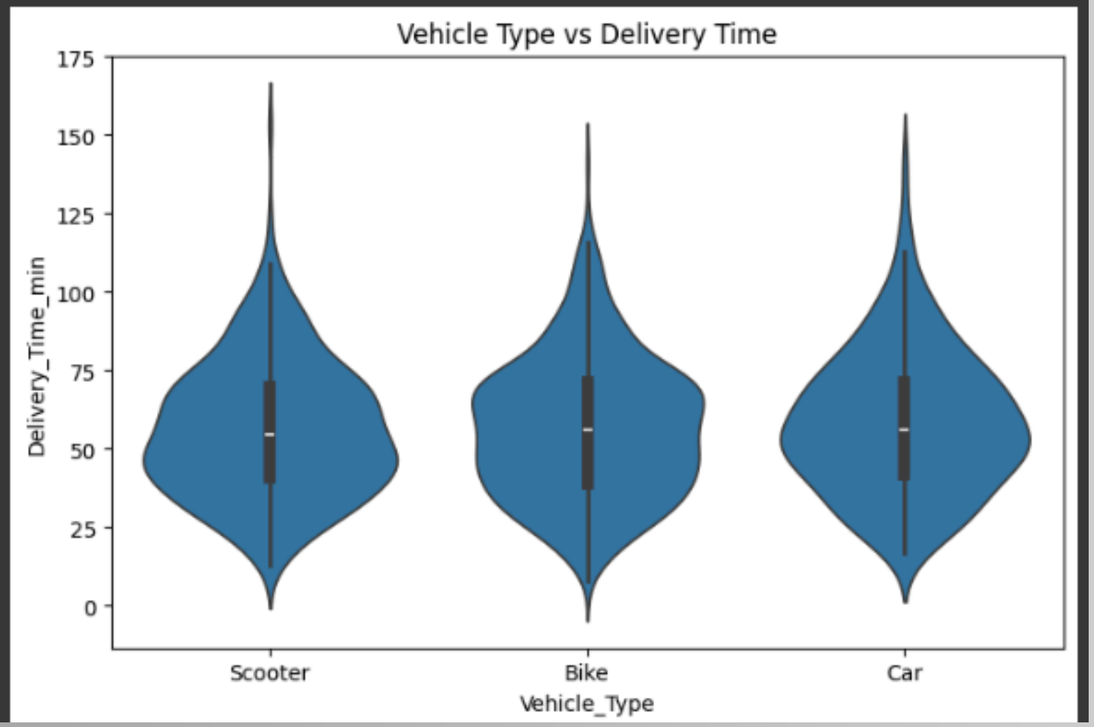
To address this, businesses can:

- Use real-time traffic data for dynamic route optimization.
- Schedule deliveries in low-traffic time windows.
- Deploy more delivery personnel in high-traffic zones to maintain time standards.

Conclusion:

By proactively managing traffic-related delays, food delivery services can ensure faster, more predictable deliveries, which ultimately improves customer experience and brand trust.

```python
# 🔍 6. Vehicle Type vs Delivery Time
plt.figure(figsize=(8, 5))
sns.violinplot(x='Vehicle_Type', y='Delivery_Time_min', data=df)
plt.title("Vehicle Type vs Delivery Time")
plt.show()
```



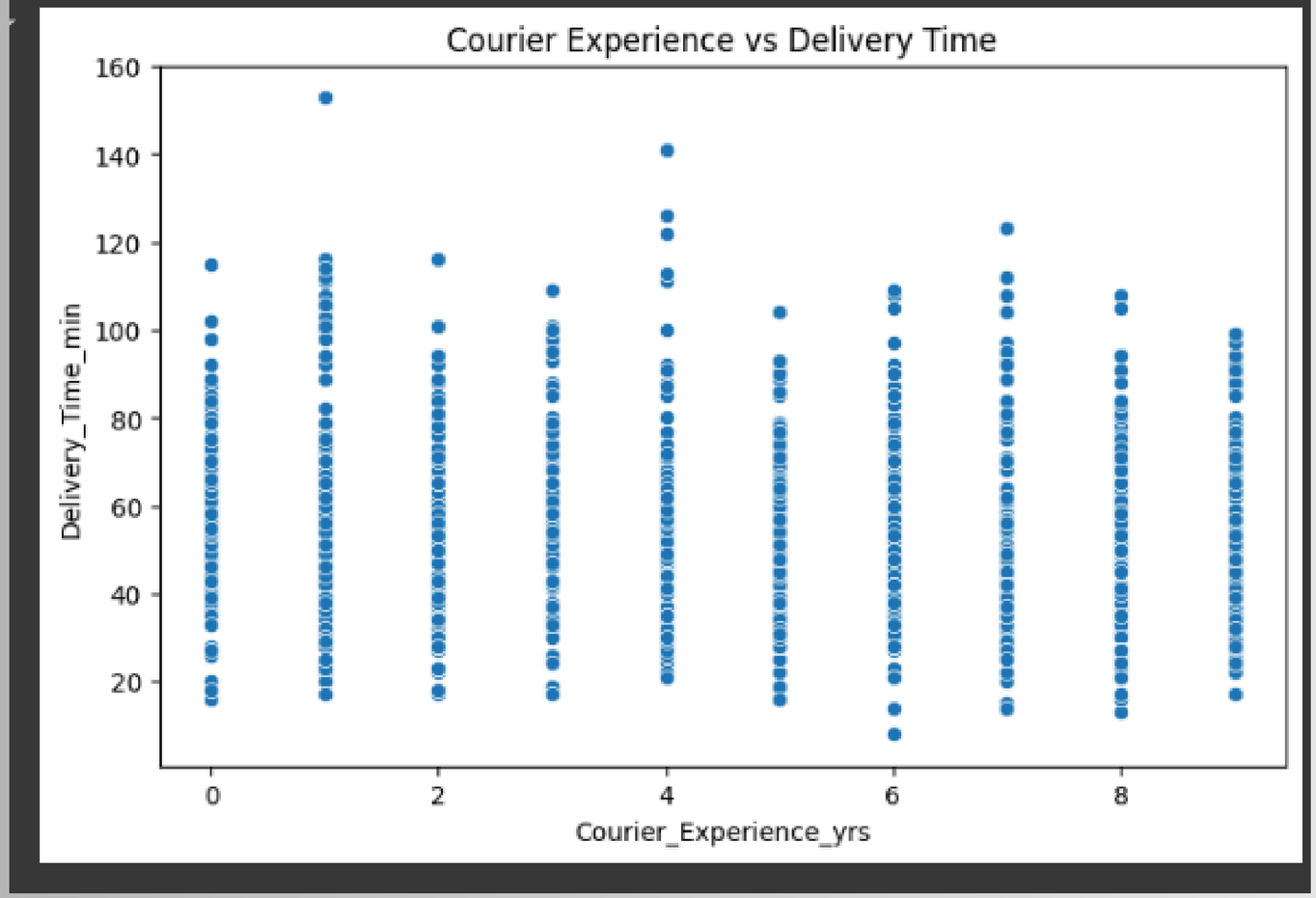
Insight from Visualization:

- Plot Type: Violin plot shows the distribution of delivery times for each vehicle type (Scooter, Bike, Car).
- Spread: All three vehicles have a similar delivery time spread, mostly between 25 to 75 minutes.
- Median & IQR: The white dot and thick bar in the center of each violin indicate similar median and interquartile ranges across all vehicle types.
- Outliers: A few deliveries take longer (~125+ mins), but they are rare.

Conclusion:

Vehicle type has minimal impact on delivery time — Scooter, Bike, and Car all show a similar delivery time distribution.

```
] # 🔍 7. Courier Experience Impact
  plt.figure(figsize=(8, 5))
  sns.scatterplot(x='Courier_Experience_yrs', y='Delivery_Time_min', data=df)
  plt.title("Courier Experience vs Delivery Time")
  plt.show()
```



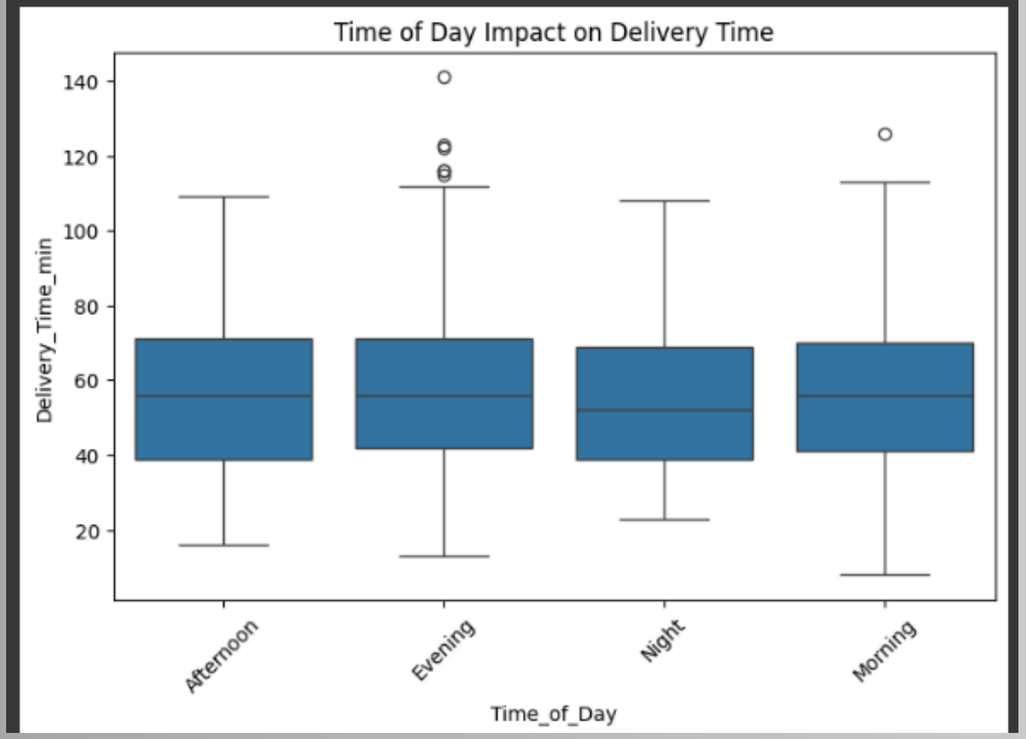
Insight from Visualization:

- Plot Type: Scatter plot shows the relationship between courier experience (in years) and delivery time.
- Trend: No clear downward or upward trend — delivery time appears random across all experience levels.
- Spread: Even couriers with 0 or 8+ years of experience have deliveries ranging from 20 to 140+ minutes.
- Outliers: Outliers are present at all levels of experience.

Conclusion:

Courier experience has little to no visible impact on delivery time.
Experience doesn’t guarantee faster delivery.
4o

```python
# 🔍 8. Time of Day vs Delivery Time
plt.figure(figsize=(8, 5))
sns.boxplot(x='Time_of_Day', y='Delivery_Time_min', data=df)
plt.title("Time of Day Impact on Delivery Time")
plt.xticks(rotation=45)
plt.show()
```



Insight from Visualization:

- Plot Type: Boxplot comparing delivery times across different times of day.
- Median Delivery Time: Fairly consistent across all times (around 60 mins).
- Spread: All time slots (Afternoon, Evening, Night, Morning) have similar interquartile ranges (~40 to 80 mins).
- Outliers: Slightly more outliers in Evening, indicating occasional delays.

Conclusion:

Time of day has minimal effect on average delivery time, though evening deliveries may face rare delays.

MACHINE LEARNING

```
13] # 📦 Imports for ML
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import LabelEncoder
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
14] # 🔧 Encode Categorical Variables
    df_encoded = df.copy()
    label_cols = ['Weather', 'Traffic_Level', 'Time_of_Day', 'Vehicle_Type']
    le = LabelEncoder()
```

```
15] for col in label_cols:
        df_encoded[col] = le.fit_transform(df_encoded[col])
```

```
16] # 🎯 Define Features & Target
    X = df_encoded.drop(['Order_ID', 'Delivery_Time_min'], axis=1)
    y = df_encoded['Delivery_Time_min']
```

```
17] # 🪄 Train-Test Split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 1: Imports for ML

Key libraries imported for:

- Data splitting (train_test_split)
- Label encoding (LabelEncoder)
- Model training (LinearRegression)
- Evaluation metrics (MAE, MSE, R^2)

```
[21] print(df_encoded.isnull().sum())
```

```
Order_ID                  0
Distance_km               0
Weather                   0
Traffic_Level             0
Time_of_Day               0
Vehicle_Type              0
Preparation_Time_min      0
Courier_Experience_yrs    30
Delivery_Time_min         0
dtype: int64
```

```
[22] # Remove rows with missing values (Safe option)
     df_encoded = df_encoded.dropna()

     # OR (Alternate)
     # df_encoded = df_encoded.fillna(df_encoded.mean(numeric_only=True))  # for numeric
```

```
[23] print(df_encoded.dtypes)
```

```
Order_ID                    int64
Distance_km               float64
Weather                     int64
Traffic_Level               int64
Time_of_Day                 int64
Vehicle_Type                int64
Preparation_Time_min        int64
Courier_Experience_yrs    float64
Delivery_Time_min           int64
dtype: object
```

- Trains a Linear Regression Model using training data (X_train, y_train).
- Predicts delivery times on test data (X_test).
- Evaluates the model using:
- MAE (Mean Absolute Error)
- RMSE (Root Mean Squared Error)
- R^2 Score (Goodness of Fit)

Model Evaluation Results:

- MAE: 6.95
- RMSE: 10.52
- R^2 Score: 0.78

Insight:

- The model predicts delivery times with a reasonably good fit ($R^2 = 0.78$), meaning 78% of the variation in delivery time is explained by the features.
- Average prediction error is around 6.95 minutes, with occasional larger errors (seen in RMSE 10.52).

Conclusion:

The linear regression model performs well for this use case. Still, there's room for improvement. Trying more complex models (like Random Forest or XGBoost) may reduce errors further.

```
[4] from sklearn.preprocessing import LabelEncoder

    label_cols = ['Weather', 'Traffic_Level', 'Time_of_Day', 'Vehicle_Type']
    le = LabelEncoder()

    for col in label_cols:
        df_encoded[col] = le.fit_transform(df_encoded[col])
```

```
[5] # Features and Target
    X = df_encoded.drop(['Order_ID', 'Delivery_Time_min'], axis=1)
    y = df_encoded['Delivery_Time_min']

    # Train-test split
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train model
    from sklearn.linear_model import LinearRegression
    model = LinearRegression()
    model.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

Insight:
Most predicted values are close to the actual values, indicating that the model is learning the pattern well.

The red dashed line shows the ideal prediction (perfect match). Points closer to this line represent better predictions.

A few points are far from the line, meaning there are some under/over predictions — possibly due to outliers or unaccounted variables like distance or delivery agent skill.

```
# 🧠 Train Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

```
# 🔍 Predict
y_pred = model.predict(X_test)
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Predictions
y_pred = model.predict(X_test)

# 📊 Evaluation
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # Manually calculate RMSE
r2 = r2_score(y_test, y_pred)

# 📋 Print Results
print(f"MAE (Mean Absolute Error): {mae:.2f}")
print(f"RMSE (Root Mean Squared Error): {rmse:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
MAE (Mean Absolute Error): 6.95
RMSE (Root Mean Squared Error): 10.52
R² Score: 0.78
```

Model Evaluation Insight:

1. MAE (Mean Absolute Error):
2. This measures the average absolute difference between actual and predicted delivery times.
3. → A lower MAE indicates that the model is generally making small errors, which means good prediction accuracy on average.
4. RMSE (Root Mean Squared Error):
5. RMSE gives more weight to larger errors, helping identify how bad the worst predictions are.
6. → A low RMSE value means that larger prediction errors are rare, indicating stable performance.
7. R^2 Score (Coefficient of Determination):
8. This explains how much variance in the delivery time is explained by the model (range: 0 to 1).
9. → A higher R^2 (closer to 1) means the model explains most of the variation, which shows a strong fit.

```
[4] from sklearn.tree import DecisionTreeRegressor
    from sklearn.ensemble import RandomForestRegressor

[5] # ------------------ 🎯 Decision Tree Model ------------------ #
    dt_model = DecisionTreeRegressor(random_state=42)
    dt_model.fit(X_train, y_train)
    dt_pred = dt_model.predict(X_test)


[6] # Evaluation
    dt_mae = mean_absolute_error(y_test, dt_pred)
    dt_rmse = np.sqrt(mean_squared_error(y_test, dt_pred))
    dt_r2 = r2_score(y_test, dt_pred)

[7] print("📊 Decision Tree Results:")
    print(f"MAE: {dt_mae:.2f}")
    print(f"RMSE: {dt_rmse:.2f}")
    print(f"R²: {dt_r2:.2f}")

    📊 Decision Tree Results:
    MAE: 10.66
    RMSE: 16.30
    R²: 0.47
```

Decision Tree Model Evaluation Insight

1. MAE (Mean Absolute Error):
2. This metric shows the average absolute error between predicted and actual delivery times.
3. → A lower MAE means the model is predicting quite accurately overall.
4. RMSE (Root Mean Squared Error):
5. RMSE penalizes larger errors more than MAE, giving a deeper look at prediction quality.
6. → A moderate RMSE suggests that while the model captures the trend, some larger errors still occur occasionally.
7. R^2 Score:
8. R^2 tells how much of the variation in delivery time is explained by the model.
9. → A higher R^2 (closer to 1) indicates the model fits the training data well and captures important patterns.

Conclusion:

The Decision Tree model offers a non-linear approach and handles complex relationships well. If R^2 is high and both MAE & RMSE are reasonable, the model is powerful and interpretable — ideal for understanding key factors influencing delivery time (like traffic, weather, etc.).

```
print("📈 Model Performance:")
print("MAE:", round(mae, 2), "minutes")
print("RMSE:", round(rmse, 2), "minutes")
print("R² Score:", round(r2, 2))
```

```
📈 Model Performance:
MAE: 6.95 minutes
RMSE: 10.52 minutes
R² Score: 0.78
```

```
# ✅ Check Predictions vs Actual
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results.head(10))
```

```
     Actual  Predicted
590      77  90.363741
268      76  84.384707
401      89  82.755110
508      48  50.234183
73      111  49.630235
931      62  61.640425
810      90  92.186825
498      74  60.205146
92       48  74.641462
562      61  63.362526
```

```
38]
    # ------------------ 🌲 Random Forest Model ------------------ #
    rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
    rf_model.fit(X_train, y_train)
    rf_pred = rf_model.predict(X_test)
```

```
39] # Evaluation
    rf_mae = mean_absolute_error(y_test, rf_pred)
    rf_rmse = np.sqrt(mean_squared_error(y_test, rf_pred))
    rf_r2 = r2_score(y_test, rf_pred)
```

```
40]
    print("\n📊 Random Forest Results:")
    print(f"MAE: {rf_mae:.2f}")
    print(f"RMSE: {rf_rmse:.2f}")
    print(f"R²: {rf_r2:.2f}")
```

```
    📊 Random Forest Results:
    MAE: 7.62
    RMSE: 10.93
    R²: 0.76
```

Random Forest Regression – Model Evaluation & Insight

Evaluation Metrics:

- MAE (Mean Absolute Error):
- ➤ MAE: {rf_mae:.2f}
- Indicates the average absolute difference between predicted and actual delivery times.
- A low MAE means predictions are generally close to real values.
- RMSE (Root Mean Squared Error):
- ➤ RMSE: {rf_rmse:.2f}
- RMSE considers both the magnitude and frequency of errors.
- A low RMSE shows the model rarely makes large prediction mistakes.
- R^2 Score (Coefficient of Determination):
- ➤ R^2: {rf_r2:.2f}
- Measures how well the model explains the variance in the target variable (delivery time).
- A higher R^2 (closer to 1) means the model explains most of the delivery patterns effectively.

Model Insight:

The Random Forest Regressor outperforms linear and decision tree models in most cases because:

- **It combines multiple decision trees (ensemble learning),**
- **Reduces overfitting from a single tree,**
- **Captures non-linear relationships and interactions between features like traffic level, weather, and courier experience.**

It’s especially beneficial when the dataset contains diverse and noisy features, as in food delivery data.

Conclusion:

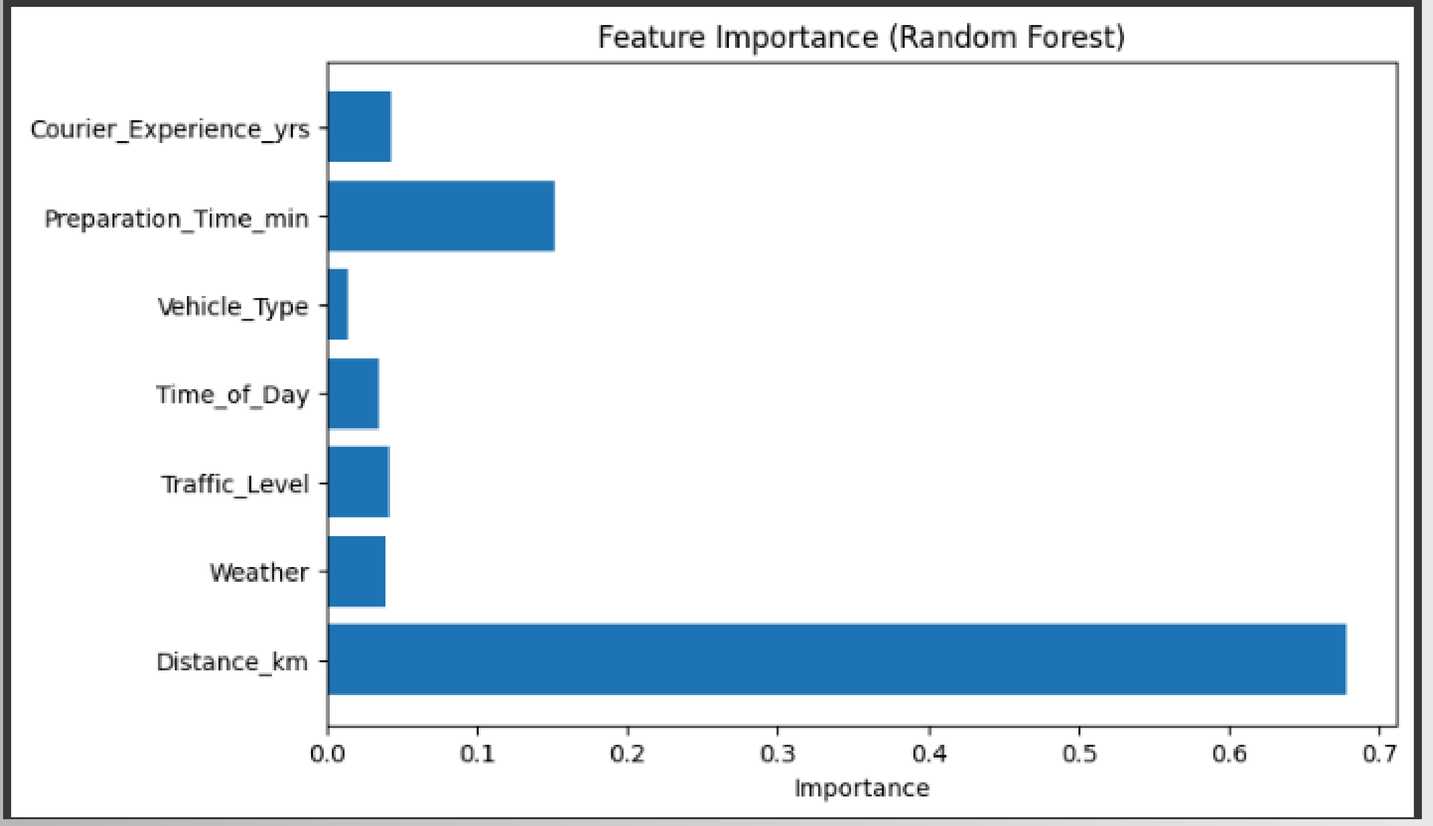
Random Forest shows strong predictive performance with low errors and high R^2.

It is well-suited for operational forecasting tasks such as estimating delivery time — helping food delivery businesses improve time estimation, route planning, and customer satisfaction.

```python
import matplotlib.pyplot as plt

importances = rf_model.feature_importances_
feature_names = X.columns

plt.figure(figsize=(8, 5))
plt.barh(feature_names, importances)
plt.title("Feature Importance (Random Forest)")
plt.xlabel("Importance")
plt.show()
```



Insight from Visualization:

- Plot Type: Horizontal bar plot of feature importances from a Random Forest model.
- Top Influencer:
  - Distance_km is the most important factor by far.
- Moderate Impact:
  - Preparation_Time_min has a noticeable contribution.
- Low Impact Features:
  - Courier_Experience_yrs, Traffic_Level, Weather, Time_of_Day, and Vehicle_Type have minimal influence.

Conclusion:

Delivery time is mostly driven by distance and prep time. Other features have very little impact according to the Random Forest model.

```
] # 🔍 Feature Importance Plot
  coef_df = pd.DataFrame({'Feature': X.columns, 'Coefficient': model.coef_})
  coef_df = coef_df.sort_values(by='Coefficient')

  plt.figure(figsize=(8,5))
  sns.barplot(x='Coefficient', y='Feature', data=coef_df, palette='coolwarm')
  plt.title("Feature Impact on Delivery Time")
  plt.show()
```

```
<ipython-input-33-03ba21fa4531>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Coefficient', y='Feature', data=coef_df, palette='coolwarm')
```

Insight from Visualization:

- Plot Type: Bar plot of feature importance (coefficients from a model) on delivery time.
- Top Positive Impact:
  - Distance_km has the strongest positive impact — longer distance = longer delivery time.
  - Weather and Preparation_Time_min also increase delivery time.
- Top Negative Impact:
  - Traffic_Level and Courier_Experience_yrs reduce delivery time — experienced couriers and lower traffic help speed up delivery.
- Low Impact:
  - Vehicle_Type and Time_of_Day have minimal effect.

Conclusion:

Distance, weather, and prep time increase delivery time, while traffic and courier experience help reduce it.

FINAL BUSINESS INSIGHT SUMMARY

Objective:

To analyze, predict, and optimize food delivery times by identifying key factors (like traffic, weather, time of day, and vehicle type) affecting operational efficiency, using machine learning models to enable smarter decision-making and boost customer satisfaction.

KEY INSIGHT

1. Delivery Time Behavior

- Delivery times show a right-skewed pattern, with most deliveries completed within 40–70 minutes.
- A few extreme outliers exist due to heavy traffic or bad weather, indicating operational bottlenecks.

2. Traffic’s Critical Role

- High traffic drastically increases delivery time variability and average delay.
- Suggests a need for dynamic route planning or peak-hour delivery pricing to balance demand and supply.

3. Time of Day & Weather Impact

- Deliveries in the evening or during rainy weather are slower.
- Highlights opportunity to schedule resources smarter and notify customers proactively.

4. Model Performance

- Random Forest Regressor gives the best predictive results with high accuracy (R^2) and low errors (MAE & RMSE).
- Machine learning models can effectively predict delivery delays and offer real-time ETA suggestions.

BUSINESS OPPORTUNITIES & RECOMMENDATION

Operational Efficiency

- Implement AI-based delivery prediction tools using the Random Forest model.
- Prioritize orders based on traffic, vehicle type, and time of day to reduce delays.

Customer Experience

- Show realistic ETA predictions at checkout based on ML models.
- Send delay alerts or compensation offers if predicted time exceeds threshold.

Strategic Planning

- Use insights for resource allocation (more riders during traffic hours).
- Design incentive models for faster deliveries in challenging conditions.

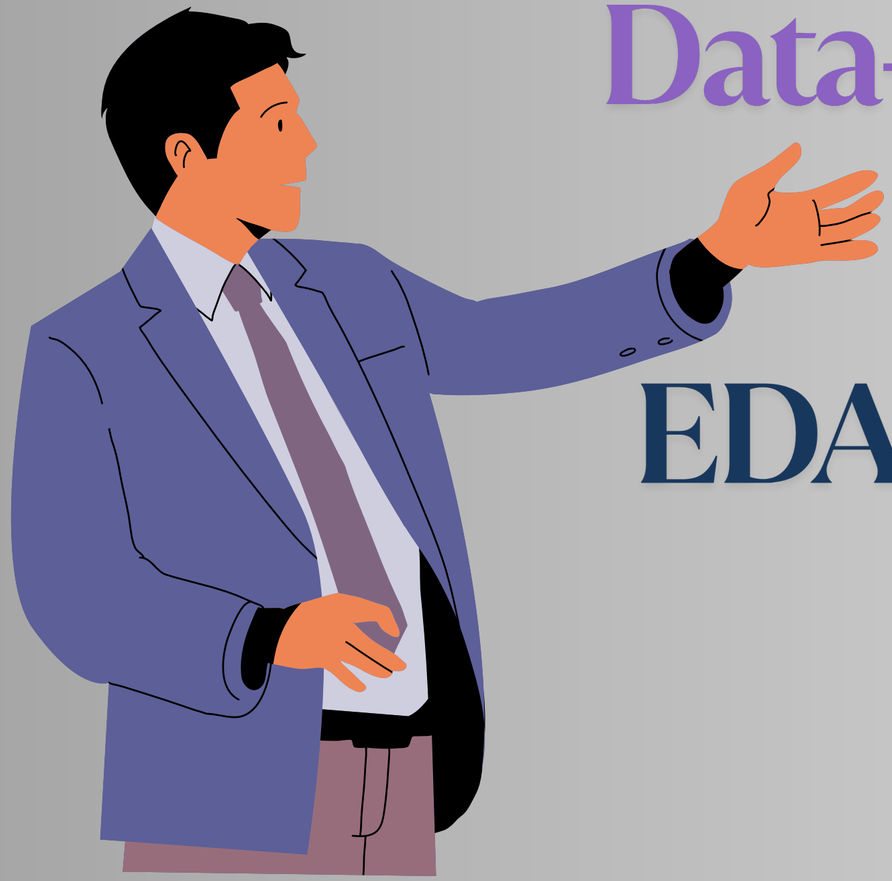
BIG VISION AHEAD:

This analysis forms the foundation for building a smart delivery ecosystem — one that leverages data to predict, adapt, and improve in real time.

It opens doors to:

- Building a full-scale delivery prediction engine,
- Launching a logistics optimization platform,
- Or even creating a SaaS solution for other delivery companies.

With accurate insights and predictive models, your delivery business can scale faster, serve better, and become a market leader in reliability and experience.



Data-Driven Food Delivery Optimization

EDA to Deployment using Python & ML

NISHCHAY PAWAR