Delivery Time Estimation using Machine Learning

Optimizing Logistics Operations Through Predictive Analytics

Capstone Project

Machine Learning Applications in Real-World Systems

Presented By:- **Abhilash Saraswat**

Why Delivery Time Prediction Matters

Customer Impact

Late deliveries cost businesses millions annually and damage customer trust, leading to negative reviews and lost revenue.

Operational Efficiency

Accurate predictions enable proactive management, better resource allocation, and improved customer satisfaction rates.

Business Value

Our project aims to forecast delivery times precisely to optimize operations and significantly reduce delivery delays.

Problem Statement

Food Delivery estimation using Machine Learning Models.

Conclusion:

Build a Machine Learning models to estimate the delivery time for food orders.

Understanding the Dataset

The dataset focuses on food delivery, a service where food is delivered to customers by restaurants, stores, or independent companies. Orders are typically placed via websites or mobile apps. The delivery can include various food items and is usually made using cars, bikes, or scooters, depending on the locations

45K

Training Records

Complete dataset with target delivery times

10K

Test Records

Dataset without target for final predictions

	ID	Delivery_person_ID	Delivery_person_Age	Delivery_person_Ratings	Restaurant_latitude	Restaurant_longitude	Delivery_location_latitude	Delivery_location_longitude	Order_Date	Time_Orderd	Time_Order_picked	Weatherconditions	Road_traffic_density	Vehicle_condition	Type_of_order	Type_of_vehicle
0	0x4607	INDORES13DEL02	37	4.9	22.745049	75.892471	22.765049	75.912471	19-03-2022	11:30:00	11:45:00	conditions Sunny	High	2	Snack	motorcycle
1	0xb379	BANGRES18DEL02	34	4.5	12.913041	77.683237	13.043041	77.813237	25-03-2022	19:45:00	19:50:00	conditions Stormy	Jam	2	Snack	scooter
2	0x5d6d	BANGRES19DEL01	23	4.4	12.914264	77.678400	12.924264	77.688400	19-03-2022	08:30:00	08:45:00	conditions Sandstorms	Low	0	Drinks	motorcycle
3	0x7a6a	COIMBRES13DEL02	38	4.7	11.003669	76.976494	11.053669	77.026494	05-04-2022	18:00:00	18:10:00	conditions Sunny	Medium	0	Buffet	motorcycle
4	0x70a2	CHENRES12DEL01	32	4.6	12.972793	80.249982	13.012793	80.289982	26-03-2022	13:30:00	13:45:00	conditions Cloudy	High	1	Snack	scooter

Table of Content

Project Workflow

- Data Loading
- Data Preprocessing
- Exploratory Data Analysis(EDA)
- Model Selection and Application
- Evaluation Metrics
- Final Insights

Data Input

Preprocessing

EDA

Model Selection

Evaluation Metrics

Final Insights

Data Collection:

```
## Loading train dataset
df = pd.read_csv(r"D:\Capstone_Project\Dataset\train.csv")
df_test = pd.read_csv(r"D:\Capstone_Project\Dataset\test.csv")
```
✓ 0.2s

- Train dataset has 45593 rows and 20 columns, 19 are features and 1 is target.
- Test dataset has 45593 rows and 17 columns are features and has no targe value.

Data Preprocessing:

```
## detecting missing values
df.isnull().mean()*100
```

✓ 0.0s

- Detecting **missing values** this block of code will give column wise missing values percentage.

```
df.duplicated().sum()
```

✓ 0.0s

```
np.int64(0)
```

- Detecting for **duplicate values**, if there will be any than that will be removed.
- But in this case there was no duplicate values.

Data Preprocessing:

```
## treating the time_taken column

df["time_taken"] = df["Time_taken(min)"].str.replace("(min)","",regex=False)
df["time_taken"]=df["time_taken"].str.strip()
df["time_taken"]=df["time_taken"].astype(int)
df=df.drop("Time_taken(min)",axis=1)
df.head(5)
```

✓ 0.0s

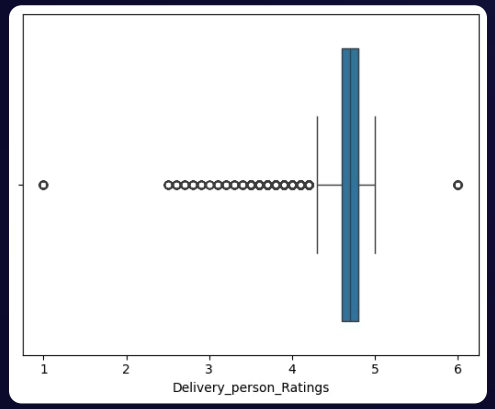
- Treating the **time_taken column** as it should be in integer format but it was object.

```
## Treating all the time columns (Order_Date, Time_Order_picked, Time_Ordered)

# converting Order_Date
df["Order_Date"] = pd.to_datetime(df["Order_Date"], format="%d-%m-%Y", errors="coerce")

# converting Time_Order_picked
df["Time_Order_picked"] = pd.to_datetime(df["Time_Order_picked"], format="%H:%M:%S", errors="coerce")

# converting Time_Ordered
df["Time_Orderd"] = pd.to_datetime(df["Time_Orderd"], format="%H:%M:%S", errors="coerce")

df["Delivery_person_Age"]=df["Delivery_person_Age"].astype(str).str.strip().replace("NaN", pd.NA) ## to remove the NaN values

df["Delivery_person_Age"]=pd.to_numeric(df["Delivery_person_Age"], errors="coerce").astype("Int64")  ## converting df to int64
```

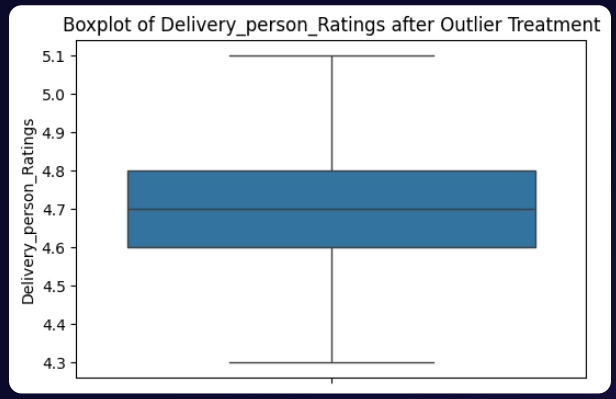
✓ 0.0s

- Treating the time columns as they are not in the datatype we are expecting.
- Converting all time based column in **date time format**.

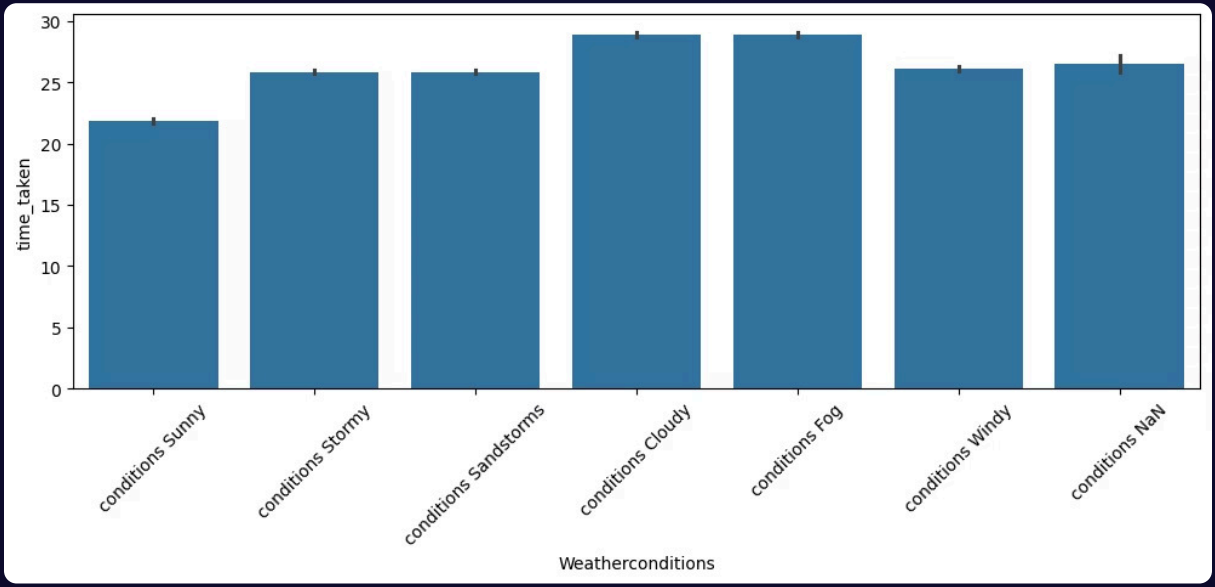
Handling Outliers:



- Detected outliers in Delivery_person_ratings.
- Used **IQR method** to handle the outliers.

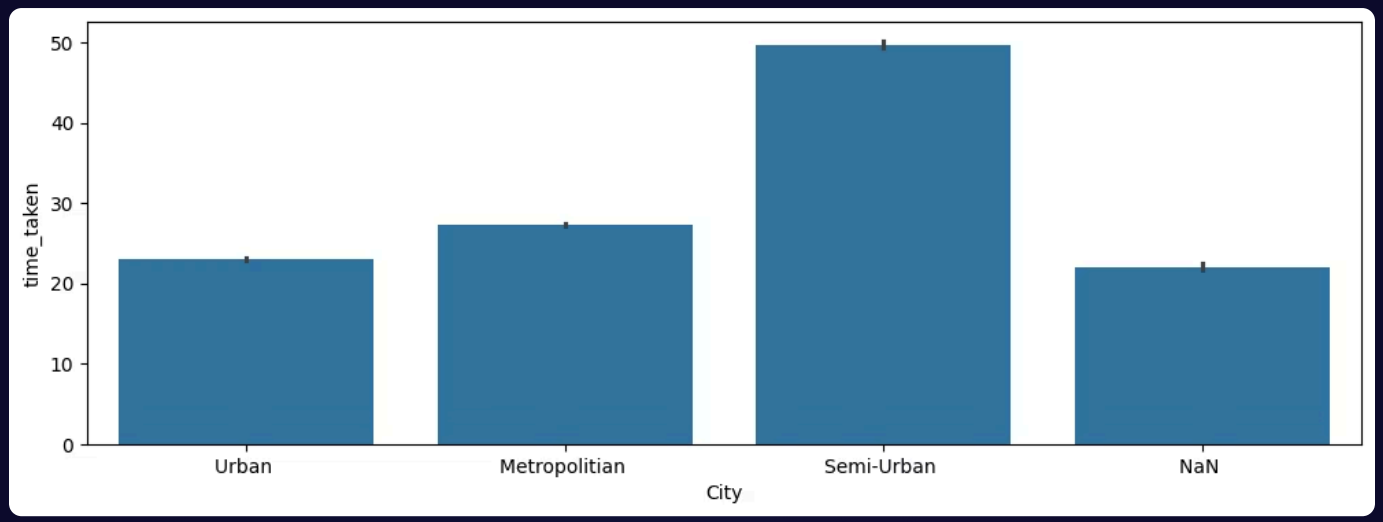


Exploratory Data Analysis (EDA)



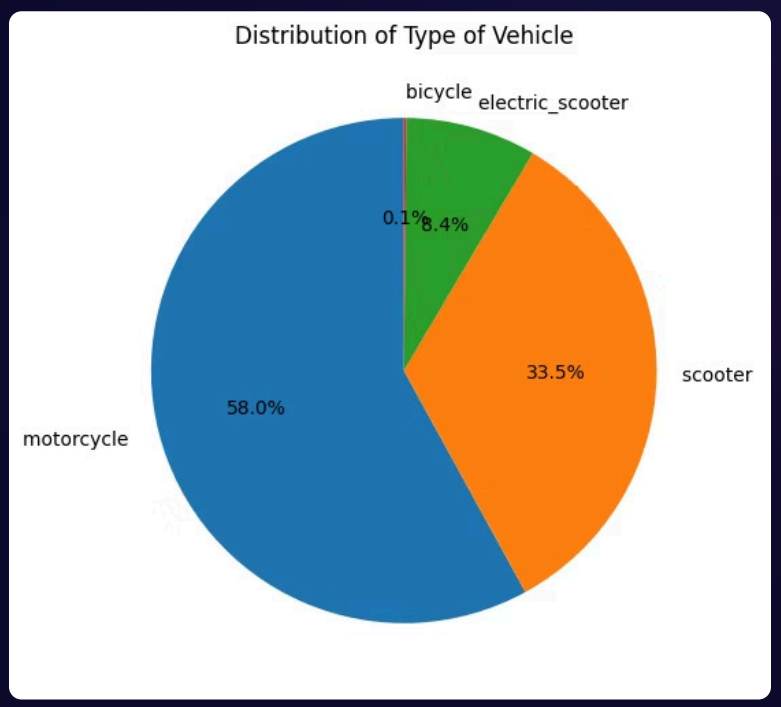
- Examined the time_taken against the Weather Conditions.
- As we can see that most time taken for delivery is when the weather is **cloudy** and **foggy**.

```
# Weatherconditions vs time_taken
plt.figure(figsize=(12,4))
sns.barplot(x="Weatherconditions", y="time_taken", data=df)
plt.xticks(rotation=45)
plt.show()
```



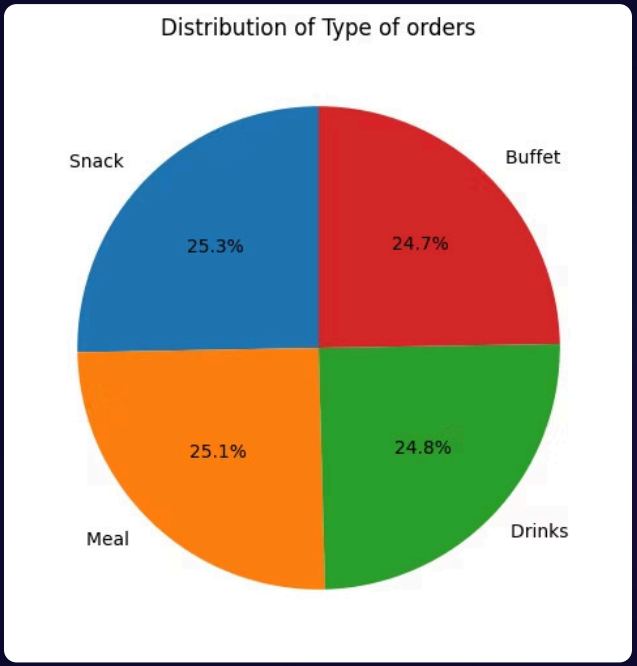
- Examined the time taken against City.
- Most time taken in semi urban cities.

```
# City vs time_taken

plt.figure(figsize=(12,4))
sns.barplot(x="City", y="time_taken", data=df)
plt.show()
```
✓ 0.8s



- Distribution of types of vehicles used by delivery person.

```
## Vehicle Distribution

plt.figure(figsize=(6,6))

# Count values of Type_of_vehicle
vehicle_counts = df["Type_of_vehicle"].value_counts()

# Plot pie chart
plt.pie(
    vehicle_counts,
    labels=vehicle_counts.index,
    autopct='%1.1f%%',
    startangle=90
)

plt.title("Distribution of Type of Vehicle")
plt.show()
```



- Distribution of type of order for delivery.
- They all have almost same distribution.

```
plt.figure(figsize=(6,6))

# Count values of Type_of_vehicle
order_counts = df["Type_of_order"].value_counts()

# Plot pie chart
plt.pie(
    order_counts,
    labels=order_counts.index,
    autopct='%1.1f%%',
    startangle=90
)

plt.title("Distribution of Type of orders")
plt.show()
```

✓ 0.1s

Train Test and Pipeline

```
## splitting the dataset into x_train and y_train
from sklearn.model_selection import train_test_split
X= df.drop(columns=["time_taken"])
Y = df["time_taken"]
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

✓ 0.3s

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
```

Model Fitting

```
## linear regression model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

lr = LinearRegression()
lr.fit(x_train.select_dtypes(include=["number"]), y_train)
```

✓ 0.2s

LinearRegression

▶ Parameters

- Baseline model. Good for identifying linear relationships between features and target.

```
## random forest regressor model
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators=100, random_state=42)
rfr.fit(x_train.select_dtypes(include=["number"]), y_train)
```

✓ 34.2s

RandomForestRegressor

▶ Parameters

- A random forest is an ensemble learning method that combines the predictions from multiple decision trees to produce a more accurate and stable prediction.

```
## xgboost regressor model
from xgboost import XGBRegressor
xgb = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb.fit(x_train.select_dtypes(include=["number"]), y_train)
```

✓ 0.3s

XGBRegressor

▶ Parameters

- XGBoost is a powerful approach for building supervised regression models. The validity of this statement can be inferred by knowing about its (XGBoost) objective function and base learners.

```python
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression

# Load test dataset
test_df = pd.read_csv(r"D:\Capstone_Project\Dataset\train.csv")

# Drop the same columns as in train preprocessing
cols_to_drop = ["Delivery_person_Ratings", "Restaurant_latitude", "Restaurant_longitude"]
test_df = test_df.drop(cols_to_drop, axis=1)

# Define numerical and categorical features
num_features = ["Delivery_person_Age", "Delivery_location_latitude", "Delivery_location_longitude"]
cat_features = ["Weatherconditions", "Road_traffic_density", "Type_of_order", "Type_of_vehicle", "Festival", "City"]

# Numeric pipeline
num_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

# Categorical pipeline
cat_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])
```

```python
# Categorical pipeline
cat_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])

# Combine into a preprocessor
preprocessor = ColumnTransformer([
    ("num", num_pipeline, num_features),
    ("cat", cat_pipeline, cat_features)
])

# Fit preprocessing on train dataset
X_train_transformed = preprocessor.fit_transform(x_train)

# Train a regressor on transformed train data
regressor = RandomForestRegressor()
regressor.fit(X_train_transformed, y_train)

# Transform test dataset
X_test_transformed = preprocessor.transform(test_df)

# Predict
y_pred_test_rfr = regressor.predict(X_test_transformed)

# Save submission
submission = pd.DataFrame({
    "ID": test_df["ID"],
    "Predicted_time_taken(min)": y_pred_test_rfr
})
print(submission)
```

	ID	Predicted_time_taken(min)
0	0x4607	21.970657
1	0xb379	33.563946
2	0x5d6d	17.269770
3	0x7a6a	23.916319
4	0x70a2	31.314754

- **Random forest Regressor** implied on test data.
- Predicted time in min for different ID's

```python
# Fit preprocessing on train dataset
X_train_transformed = preprocessor.fit_transform(x_train)

# Train a regressor on transformed train data
regressor = LinearRegression()
regressor.fit(X_train_transformed, y_train)

# Transform test dataset
X_test_transformed = preprocessor.transform(test_df)

# Predict
y_pred_test_lr = regressor.predict(X_test_transformed)

# Save submission
submission = pd.DataFrame({
    "ID": test_df["ID"],
    "Predicted_time_taken(min)": y_pred_test_lr
})
print(submission)
```

✓ 0.7s

- **Linear Regressor** implied on test dataset

	ID	Predicted_time_taken(min)
0	0x4607	25.288602
1	0xb379	31.034373
2	0x5d6d	17.255120
3	0x7a6a	27.874378
4	0x70a2	29.331190

- Gives better predictions in some cases and give a bit dull predictions in some cases.

```python
# Combine into a preprocessor
preprocessor = ColumnTransformer([
    ("num", num_pipeline, num_features),
    ("cat", cat_pipeline, cat_features)
])

# Fit preprocessing on train dataset
X_train_transformed = preprocessor.fit_transform(x_train)

# Train a regressor on transformed train data
regressor = XGBRegressor()
regressor.fit(X_train_transformed, y_train)

# Transform test dataset
X_test_transformed = preprocessor.transform(test_df)

# Predict
y_pred_test_xgb = regressor.predict(X_test_transformed)

# Save submission
submission = pd.DataFrame({
    "ID": test_df["ID"],
    "Predicted_time_taken(min)": y_pred_test_xgb
})
print(submission)
```

- **XGB Regressor** implied on test dataset.

	ID	Predicted_time_taken(min)
0	0x4607	21.970657
1	0xb379	33.563946
2	0x5d6d	17.269770
3	0x7a6a	23.916319
4	0x70a2	31.314754

- Gives similar prediction as **Random Forest Regressor**.

Metrics Scores

```
y_pred = lr.predict(x_test.select_dtypes(include=["number"]))
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"R2_Score: {r2}")
```

✓ 0.0s

```
MAE: 6.563700023238368
MSE: 67.3303557614811
RMSE: 8.205507648005765
R2_Score: 0.2320736015439775
```

- Metrics scores for **Linear Model -**Mean Absolute Error (MAE), Mean Squared Error (MSE), Right Mean Squared Error (RMSE) and R^2 Score.
- MAE: 6.563700023238368
- MSE: 67.3303557614811
- RMSE: 8.205507648005765
- R2_Score: 0.2320736015439775

```
y_pred_rfr = rfr.predict(x_test.select_dtypes(include=["number"]))
mae_rfr = mean_absolute_error(y_test, y_pred)
mse_rfr = mean_squared_error(y_test, y_pred)
rmse_rfr = np.sqrt(mse)
r2_rfr = r2_score(y_test, y_pred_rfr)
print(f"MAE: {mae_rfr}")
print(f"MSE: {mse_rfr}")
print(f"RMSE: {rmse_rfr}")
print(f"R2_Score: {r2_rfr}")
```

✓ 0.4s

```
MAE: 6.563700023238368
MSE: 67.3303557614811
RMSE: 8.205507648005765
R2_Score: 0.35817718962631684
```

- Metrics scores for **Random Forest Model -**Mean Absolute Error (MAE), Mean Squared Error (MSE), Right Mean Squared Error (RMSE) and R^2 Score.
- MAE: 6.563700023238368
- MSE: 67.3303557614811
- RMSE: 8.205507648005765
- R2_Score: 0.35817718962631684

```
y_pred_xgb = xgb.predict(x_test.select_dtypes(include=["number"]))
mae_xgb = mean_absolute_error(y_test, y_pred)
mse_xgb = mean_squared_error(y_test, y_pred)
rmse_xgb = np.sqrt(mse)
r2_xgb = r2_score(y_test, y_pred_xgb)
print(f"MAE: {mae_xgb}")
print(f"MSE: {mse_xgb}")
print(f"RMSE: {rmse_xgb}")
print(f"R2_Score: {r2_xgb}")
```

✓ 0.0s

```
MAE: 6.563700023238368
MSE: 67.3303557614811
RMSE: 8.205507648005765
R2_Score: 0.43445461988449097
```

- Metrics scores for **XG Boost Model -**Mean Absolute Error (MAE), Mean Squared Error (MSE), Right Mean Squared Error (RMSE) and R^2 Score.
- MAE: 6.563700023238368
- MSE: 67.3303557614811
- RMSE: 8.205507648005765
- R2_Score: 0.43445461988449097

Observation and Recommendation

Operations Management

Allows logistics teams to prioritize high-risk deliveries and optimize resource allocation for maximum efficiency.

Cost Reduction

Reduces operational costs by minimizing late deliveries, improving route planning, and decrease impact of weather on delivery.

Vehicle Type

Encourage or prioritize the use of the most efficient vehicle type for congested urban routes.

Order Picked Time

Reduce "wait time" for couriers and improve overall order cycle time.

Challenges & Future Enhancements

Technical Challenges

- Handling string-to-float conversion errors in preprocessing
- Managing NaN values and inconsistent data formats
- Pipeline optimization vs manual preprocessing trade-offs

Future Improvements

- Advanced hyperparameter tuning with grid search
- Integration of real-time traffic and weather data
- Deep learning models for complex pattern recognition

Project Success & Next Steps

Key Achievements

Successfully developed a robust delivery time prediction model, providing actionable insights for logistics optimization.

Future Vision

Integration with live tracking systems and expansion to comprehensive logistics intelligence platform.

Thank You!