In [1]:
```python
import pandas as pd
import numpy as np
import random
from datetime import datetime, timedelta, timezone
```

In [3]:
```python
# CONFIGURATION
n_days = 7
total_orders = 100_000
orders_per_day = total_orders // n_days
hours_range = list(range(6, 24))  # 6 AM to 11 PM

markets = [
    "New York", "Los Angeles", "Chicago", "Houston", "Phoenix",
    "Philadelphia", "Boston", "San Diego", "Dallas", "San Jose"
]

categories = [
    "pizza", "chinese", "indian", "mexican", "burgers", "sushi", "thai", "italian",
    "mediterranean", "korean", "bbq", "vegan", "seafood", "breakfast", "dessert",
    "fast food", "greek", "sandwiches", "halal", "japanese", "american", "latin",
    "noodles", "salads", "comfort food", "coffee", "southern", "bakery", "bubble te
]

store_types = ["restaurant", "grocery"]
protocols = [1, 2, 3, 4]

avg_drive_time = {
    "New York": 800, "Los Angeles": 1000, "Chicago": 850, "Houston": 950,
    "Phoenix": 750, "Philadelphia": 780, "Boston": 730, "San Diego": 790,
    "Dallas": 900, "San Jose": 820
}

avg_order_place_time = 300  # in seconds
```

In [5]:
```python
import requests

weather_cache = {}

def get_weather(city, date):
    API_KEY = "db31e747482e5ab721572b5456c70e5c"  # Replace with your key
    key = f"{city}_{date.strftime('%Y-%m-%d')}"

    if key in weather_cache:
        return weather_cache[key]

    try:
        url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={API
        response = requests.get(url)
        data = response.json()
        weather_info = {
            "weather_main": data["weather"][0]["main"],
            "temp_c": round(data["main"]["temp"] - 273.15, 1),
            "humidity": data["main"]["humidity"],
            "wind_speed": data["wind"]["speed"]
```

```
        }
    except:
        weather_info = {
            "weather_main": "Unknown",
            "temp_c": np.nan,
            "humidity": np.nan,
            "wind_speed": np.nan
        }

    weather_cache[key] = weather_info
    return weather_info
```

In [7]:

```
# Generator function
def generate_week_of_orders():
    all_data = []
    start_date = datetime(2023, 4, 1, tzinfo=timezone.utc)

    for day in range(n_days):
        for _ in range(orders_per_day):
            market = random.choice(markets)
            category = random.choice(categories)
            store_type = random.choices(store_types, weights=[0.85, 0.15])[0]  # 85

            hour = random.choice(hours_range)
            minute = random.randint(0, 59)
            second = random.randint(0, 59)
            created_at = start_date + timedelta(days=day, hours=hour, minutes=minut

            total_items = max(1, np.random.poisson(lam=3 if store_type == "restaura
            subtotal = max(800, int(np.random.normal(loc=2500 if store_type == "res
            num_distinct_items = np.random.randint(1, total_items + 1)
            min_item_price = int(subtotal / total_items * np.random.uniform(0.5, 0.
            max_item_price = int(subtotal / total_items * np.random.uniform(1.1, 1.

            total_onshift = np.random.randint(10, 50)
            total_busy = np.random.randint(0, total_onshift)
            outstanding_orders = np.random.randint(5, 25)
            busy_ratio = total_busy / total_onshift if total_onshift else 0
            stress_level = outstanding_orders / (total_onshift - total_busy + 1)

            if store_type == "grocery":
                est_order_place = avg_order_place_time + 30 * total_items
            else:
                est_order_place = avg_order_place_time + np.random.randint(-60, 90)

            est_drive_time = avg_drive_time[market] + np.random.randint(-100, 150)
            eta = est_order_place + est_drive_time + int(300 * stress_level)

            # 🌧️ Fetch real weather data
            weather = get_weather(market, created_at)

            all_data.append({
                "created_at": created_at,
                "market": market,
                "store_category": category,
                "store_type": store_type,
```

```
                "order_protocol": random.choice(protocols),
                "total_items": total_items,
                "subtotal": subtotal,
                "num_distinct_items": num_distinct_items,
                "min_item_price": min_item_price,
                "max_item_price": max_item_price,
                "total_onshift_dashers": total_onshift,
                "total_busy_dashers": total_busy,
                "total_outstanding_orders": outstanding_orders,
                "busy_dashers_ratio": round(busy_ratio, 2),
                "dasher_stress_index": round(stress_level, 2),
                "estimated_order_place_duration": est_order_place,
                "estimated_store_to_consumer_driving_duration": est_drive_time,
                "predicted_delivery_duration": eta,
                "weather_main": weather["weather_main"],
                "temp_c": weather["temp_c"],
                "humidity": weather["humidity"],
                "wind_speed": weather["wind_speed"]
            })

    return pd.DataFrame(all_data)

# -------------------- RUN & SAVE --------------------
```

In [9]:

```
# Generate the dataset
df = generate_week_of_orders()

# Enrich time-based + operational metrics
df["day_of_week"] = df["created_at"].dt.day_name()
df["hour_of_day"] = df["created_at"].dt.hour
df["is_peak_hour"] = df["hour_of_day"].apply(lambda x: 1 if x in [11,12,13,18,19,20
df["dasher_to_order_ratio"] = (df["total_onshift_dashers"] - df["total_busy_dashers
df["dasher_to_order_ratio"] = df["dasher_to_order_ratio"].replace([np.inf, -np.inf]
df["surge_flag"] = df["dasher_stress_index"].apply(lambda x: 1 if x > 2 else 0)

# Save to file
df.to_csv("simulated_doordash_100k_orders.csv", index=False)
print("✅ Generated and saved: simulated_doordash_100k_orders.csv with weather data
```

```
✅ Generated and saved: simulated_doordash_100k_orders.csv with weather data 🌦️
```

In [10]:

```
df_final = pd.read_csv(r'C:\Users\Parth Badani\Downloads\doordash_100k_dashboard_re
```

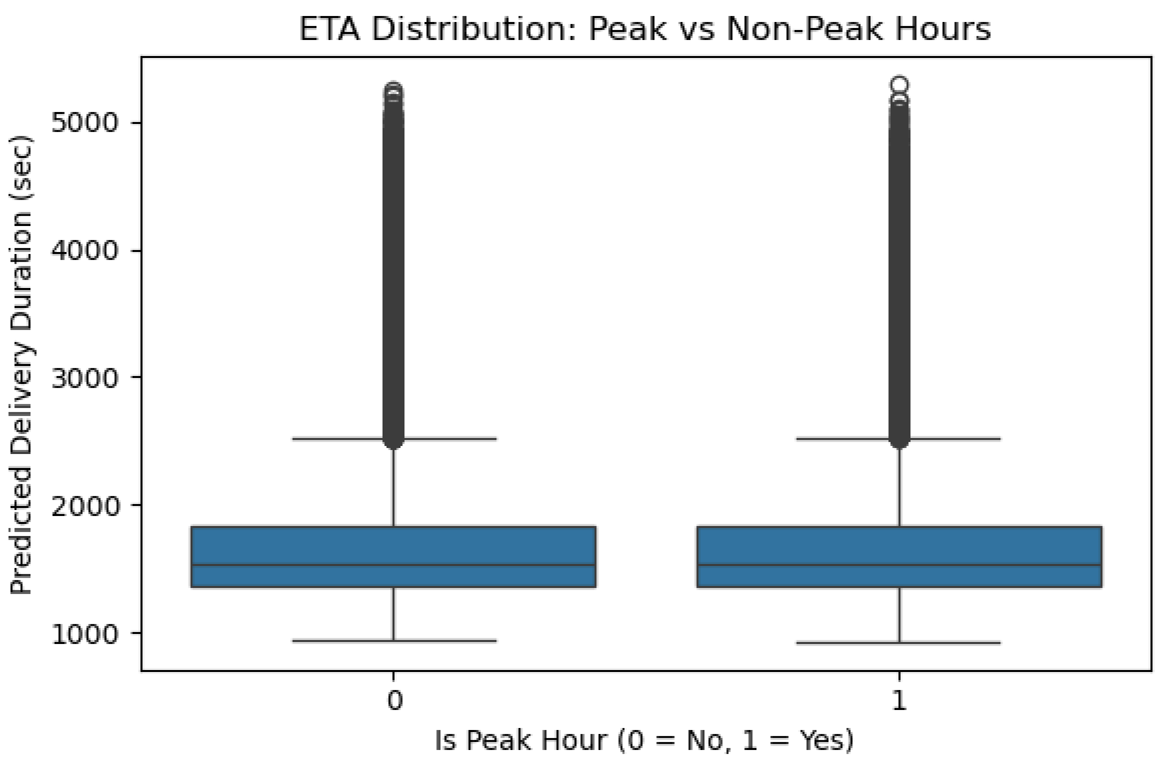
In [13]:

```
from scipy.stats import ttest_ind

peak_eta = df_final[df_final["IS_PEAK_HOUR"] == 1]["PREDICTED_DELIVERY_DURATION"]
nonpeak_eta = df_final[df_final["IS_PEAK_HOUR"] == 0]["PREDICTED_DELIVERY_DURATION"

t_stat, p_val = ttest_ind(peak_eta, nonpeak_eta, equal_var=False)

print("T-Test: Peak vs Non-Peak")
print(f"T-statistic: {t_stat:.3f}")
print(f"P-value: {p_val:.4f}")
```

```
T-Test: Peak vs Non-Peak
T-statistic: -0.638
P-value: 0.5236
```

In [15]:

```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4))
sns.boxplot(x="IS_PEAK_HOUR", y="PREDICTED_DELIVERY_DURATION", data=df_final)
plt.title("ETA Distribution: Peak vs Non-Peak Hours")
plt.xlabel("Is Peak Hour (0 = No, 1 = Yes)")
plt.ylabel("Predicted Delivery Duration (sec)")
plt.tight_layout()
plt.show()
```

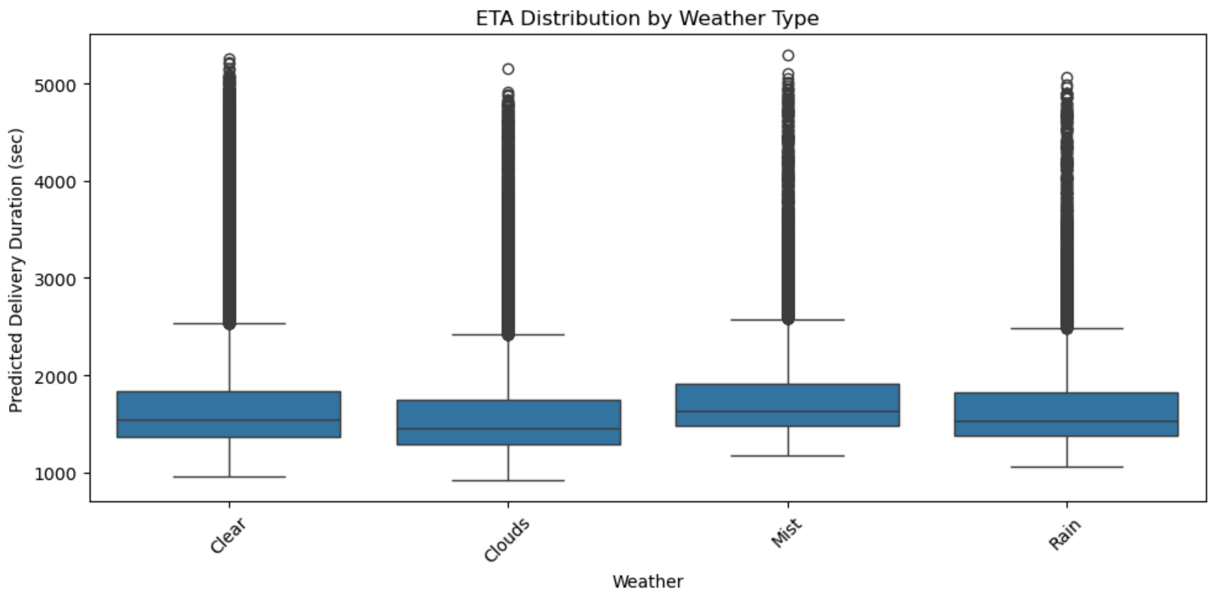


In [17]:

```python
from scipy.stats import f_oneway

weather_groups = [
    group["PREDICTED_DELIVERY_DURATION"].values
    for _, group in df_final.groupby("WEATHER_MAIN")
]

f_stat, p_anova = f_oneway(*weather_groups)

print("\nANOVA: ETA by Weather Type")
print(f"F-statistic: {f_stat:.2f}")
print(f"P-value: {p_anova:.5f}")
```

```
ANOVA: ETA by Weather Type
F-statistic: 257.18
P-value: 0.00000
```

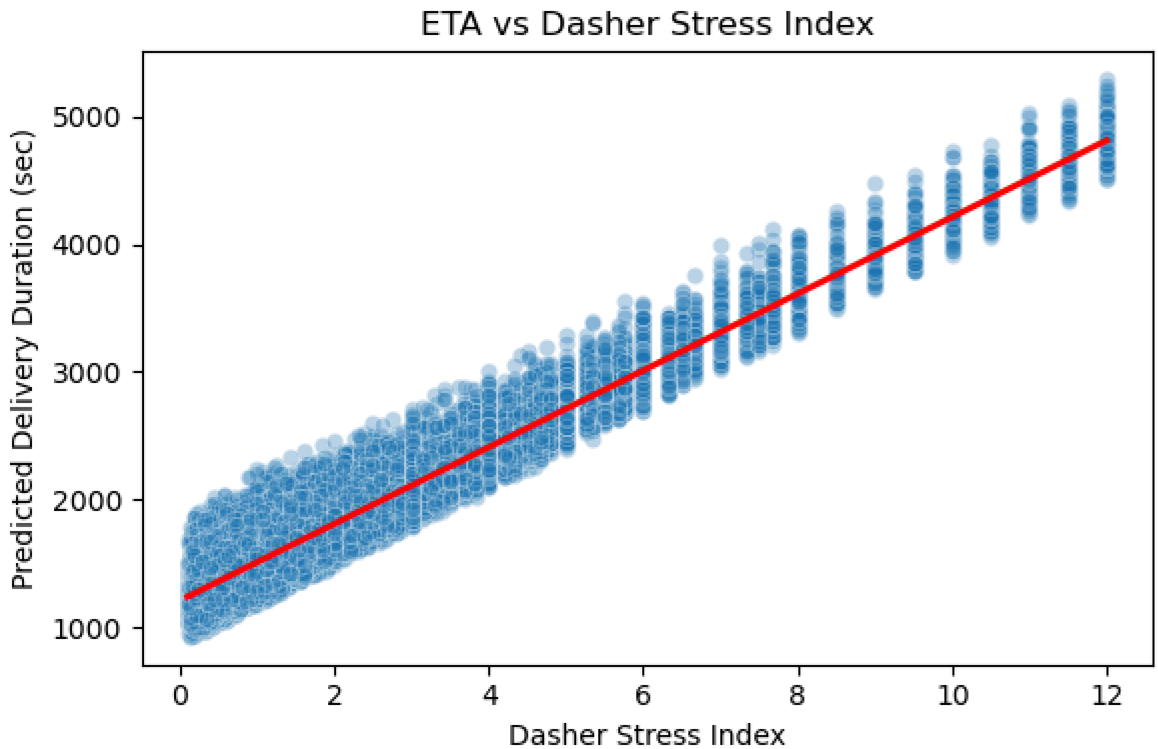
In [19]:
```
plt.figure(figsize=(10, 5))
sns.boxplot(x="WEATHER_MAIN", y="PREDICTED_DELIVERY_DURATION", data=df_final)
plt.title("ETA Distribution by Weather Type")
plt.xlabel("Weather")
plt.ylabel("Predicted Delivery Duration (sec)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [21]:
```
from scipy.stats import pearsonr

corr_val, p_corr = pearsonr(
    df_final["PREDICTED_DELIVERY_DURATION"],
    df_final["DASHER_STRESS_INDEX"]
)

print("\nPearson Correlation: ETA vs Dasher Stress Index")
print(f"Correlation Coefficient: {corr_val:.2f}")
print(f"P-value: {p_corr:.5f}")
```

```
Pearson Correlation: ETA vs Dasher Stress Index
Correlation Coefficient: 0.97
P-value: 0.00000
```

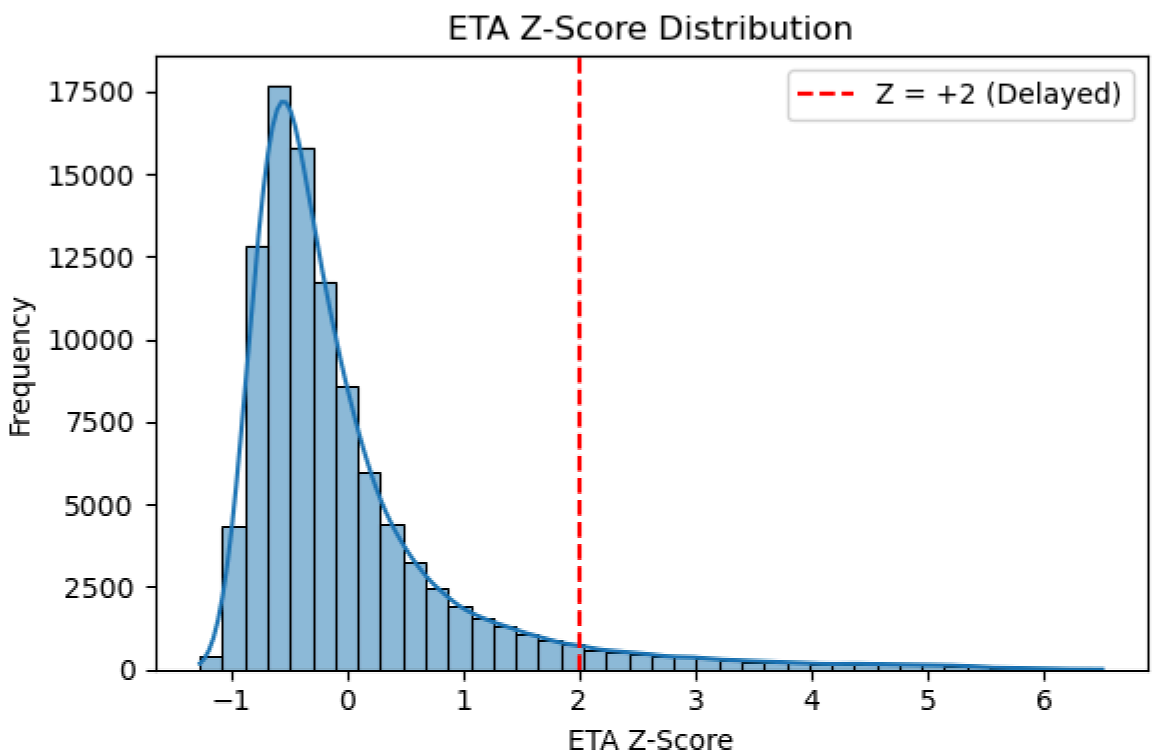
In [23]:
```
plt.figure(figsize=(6, 4))
sns.scatterplot(x="DASHER_STRESS_INDEX", y="PREDICTED_DELIVERY_DURATION", data=df_f
sns.regplot(x="DASHER_STRESS_INDEX", y="PREDICTED_DELIVERY_DURATION", data=df_final
plt.title("ETA vs Dasher Stress Index")
plt.xlabel("Dasher Stress Index")
plt.ylabel("Predicted Delivery Duration (sec)")
plt.tight_layout()
plt.show()
```



In [25]:
```
zscore_outliers_pct = (df_final["ETA_ZSCORE"] > 2).mean() * 100

print("\nZ-Score Outliers")
print(f"Percent of orders with ETA Z-Score > +2σ: {zscore_outliers_pct:.2f}%")
```

```
Z-Score Outliers
Percent of orders with ETA Z-Score > +2σ: 5.20%
```

In [27]:
```
plt.figure(figsize=(6, 4))
sns.histplot(df_final["ETA_ZSCORE"], bins=40, kde=True)
plt.axvline(2, color='red', linestyle='--', label='Z = +2 (Delayed)')
plt.title("ETA Z-Score Distribution")
plt.xlabel("ETA Z-Score")
plt.ylabel("Frequency")
plt.legend()
plt.tight_layout()
plt.show()
```



In [29]:
```
!pip install lightgbm
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: lightgbm in c:\users\parth badani\appdata\roaming\pyt
hon\python312\site-packages (4.6.0)
Requirement already satisfied: numpy>=1.17.0 in c:\programdata\anaconda3\lib\site-pa
ckages (from lightgbm) (1.26.4)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages
(from lightgbm) (1.13.1)
```

In [31]:
```
import sys
!{sys.executable} -m pip install lightgbm
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: lightgbm in c:\users\parth badani\appdata\roaming\pyt
hon\python312\site-packages (4.6.0)
Requirement already satisfied: numpy>=1.17.0 in c:\programdata\anaconda3\lib\site-pa
ckages (from lightgbm) (1.26.4)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages
(from lightgbm) (1.13.1)
```

In [33]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_squared_error
```

In [35]:
```
# Load your dataset
df_final = pd.read_csv(r'C:\Users\Parth Badani\Downloads\doordash_100k_dashboard_re

# Define target and remove non-features
target = "PREDICTED_DELIVERY_DURATION"
```

```python
drop_cols = [
    "PREDICTED_DELIVERY_DURATION", "ETA_ZSCORE", "CREATED_AT", "DAY_OF_WEEK",
    "MARKET", "WEATHER_MAIN", "STORE_CATEGORY"
]
```

In [37]:

```python
# Optional: Drop high-cardinality categoricals or one-hot encode
df_model = pd.get_dummies(df_final.drop(columns=drop_cols), drop_first=True)
```

In [39]:

```python
# Setup X, y
X = df_model
y = df_final[target]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [41]:

```python
# Train LightGBM
model = LGBMRegressor(random_state=42)
model.fit(X_train, y_train)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing wa
s 0.016308 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2354
[LightGBM] [Info] Number of data points in the train set: 79996, number of used feat
ures: 23
[LightGBM] [Info] Start training from score 1699.373369
```

Out[41]:

```
▾ LGBMRegressor
LGBMRegressor(random_state=42)
```

In [45]:

```python
# Predict and Evaluate
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"✅ RMSE: {rmse:.2f} seconds")
```

```
✅ RMSE: 8.95 seconds
```

In [47]:

```python
# Feature Importances
importances = pd.DataFrame({
    "Feature": X.columns,
    "Importance": model.feature_importances_
}).sort_values(by="Importance", ascending=False)

print("\n🔍 Top 15 Features:")
print(importances.head(15))
```

```
🔍 Top 15 Features:
                                  Feature  Importance
12  ESTIMATED_STORE_TO_CONSUMER_DRIVING_DURATION        1015
10                        DASHER_STRESS_INDEX         798
11               ESTIMATED_ORDER_PLACE_DURATION         720
1                                 TOTAL_ITEMS         194
18                        DASHER_TO_ORDER_RATIO         190
8                      TOTAL_OUTSTANDING_ORDERS          58
2                                    SUBTOTAL          12
5                              MAX_ITEM_PRICE           3
15                                 WIND_SPEED           3
4                              MIN_ITEM_PRICE           2
9                          BUSY_DASHERS_RATIO           2
21                                    STD_ETA           1
6                        TOTAL_ONSHIFT_DASHERS           1
20                                    AVG_ETA           1
17                               IS_PEAK_HOUR           0
```

In [49]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_squared_error
```

In [51]:
```python
# One-hot encode WEATHER_MAIN
weather_dummies = pd.get_dummies(df_final["WEATHER_MAIN"], prefix="WEATHER", drop_f
df_enhanced = pd.concat([df_final.copy(), weather_dummies], axis=1)
```

In [53]:
```python
# Drop non-features and leakage columns
drop_cols_enhanced = [
    "PREDICTED_DELIVERY_DURATION", "ETA_ZSCORE", "CREATED_AT", "DAY_OF_WEEK",
    "MARKET", "WEATHER_MAIN", "STORE_CATEGORY"
]
```

In [55]:
```python
# Setup X, y
X_enhanced = pd.get_dummies(df_enhanced.drop(columns=drop_cols_enhanced), drop_firs
y_enhanced = df_enhanced["PREDICTED_DELIVERY_DURATION"]
```

In [57]:
```python
# Train-test split
X_train_enh, X_test_enh, y_train_enh, y_test_enh = train_test_split(X_enhanced, y_e
```

In [59]:
```python
# Retrain LightGBM
model_enhanced = LGBMRegressor(random_state=42)
model_enhanced.fit(X_train_enh, y_train_enh)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing wa
s 0.013818 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2360
[LightGBM] [Info] Number of data points in the train set: 79996, number of used feat
ures: 26
[LightGBM] [Info] Start training from score 1699.373369
```

Out[59]:

```
LGBMRegressor
LGBMRegressor(random_state=42)
```

In [63]:

```python
# Predict and evaluate
y_pred_enh = model_enhanced.predict(X_test_enh)
rmse_enh = np.sqrt(mean_squared_error(y_test_enh, y_pred_enh))
print(f"✅ RMSE (Weather-Enhanced Model): {rmse_enh:.2f} seconds")
```

```
✅ RMSE (Weather-Enhanced Model): 8.95 seconds
```

In [65]:

```python
# Feature importances
importances_enh = pd.DataFrame({
    "Feature": X_enhanced.columns,
    "Importance": model_enhanced.feature_importances_
}).sort_values(by="Importance", ascending=False)

print("\n🔍 Top 15 Features:")
print(importances_enh.head(15))
```

```
🔍 Top 15 Features:
                                 Feature  Importance
12  ESTIMATED_STORE_TO_CONSUMER_DRIVING_DURATION        1015
10                     DASHER_STRESS_INDEX         798
11             ESTIMATED_ORDER_PLACE_DURATION         720
1                              TOTAL_ITEMS         194
18                   DASHER_TO_ORDER_RATIO         190
8                  TOTAL_OUTSTANDING_ORDERS          58
2                                 SUBTOTAL          12
15                              WIND_SPEED           3
5                           MAX_ITEM_PRICE           3
9                       BUSY_DASHERS_RATIO           2
4                           MIN_ITEM_PRICE           2
6                     TOTAL_ONSHIFT_DASHERS           1
21                                 STD_ETA           1
20                                 AVG_ETA           1
19                              SURGE_FLAG           0
```

In [67]:

```python
df_inter = df_enhanced.copy()
```

In [69]:

```python
# Interaction: peak × weather
df_inter["PEAK_RAIN"] = df_inter["IS_PEAK_HOUR"] * df_inter.get("WEATHER_Rain", 0)
df_inter["PEAK_SNOW"] = df_inter["IS_PEAK_HOUR"] * df_inter.get("WEATHER_Snow", 0)
```

In [71]:

```python
# Interaction: grocery × hour (you can adjust this logic if you’ve one-hot encoded
df_inter["IS_GROCERY"] = df_inter["STORE_TYPE"].apply(lambda x: 1 if x == "grocery"
df_inter["GROCERY_HOUR"] = df_inter["HOUR_OF_DAY"] * df_inter["IS_GROCERY"]
df_inter["GROCERY_ITEMS"] = df_inter["TOTAL_ITEMS"] * df_inter["IS_GROCERY"]
```

In [73]:

```python
# Final drop list
drop_cols_final = [
    "PREDICTED_DELIVERY_DURATION", "ETA_ZSCORE", "CREATED_AT", "DAY_OF_WEEK",
    "MARKET", "WEATHER_MAIN", "STORE_CATEGORY"
]
```

```python
X_inter = pd.get_dummies(df_inter.drop(columns=drop_cols_final), drop_first=True)
y_inter = df_inter["PREDICTED_DELIVERY_DURATION"]
```

In [75]:
```python
# Train-test split
X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(X_inter, y_inter, test_
```

In [77]:
```python
# LightGBM with interaction terms
model_i = LGBMRegressor(random_state=42)
model_i.fit(X_train_i, y_train_i)
y_pred_i = model_i.predict(X_test_i)
rmse_i = np.sqrt(mean_squared_error(y_test_i, y_pred_i))
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.045313 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2405
[LightGBM] [Info] Number of data points in the train set: 79996, number of used feat
ures: 30
[LightGBM] [Info] Start training from score 1699.373369
```

In [79]:
```python
# Feature importances
importances_i = pd.DataFrame({
    "Feature": X_inter.columns,
    "Importance": model_i.feature_importances_
}).sort_values(by="Importance", ascending=False)

print(f"✅ RMSE (with interaction terms): {rmse_i:.2f} seconds")
print("\n🔍 Top 15 Features:")
print(importances_i.head(15))
```

```
✅ RMSE (with interaction terms): 8.96 seconds

🔍 Top 15 Features:
                                    Feature  Importance
12  ESTIMATED_STORE_TO_CONSUMER_DRIVING_DURATION        1010
10                          DASHER_STRESS_INDEX         795
11                ESTIMATED_ORDER_PLACE_DURATION         736
1                                   TOTAL_ITEMS         192
18                           DASHER_TO_ORDER_RATIO         180
8                       TOTAL_OUTSTANDING_ORDERS          64
2                                      SUBTOTAL           7
5                                MAX_ITEM_PRICE           4
9                            BUSY_DASHERS_RATIO           3
15                                   WIND_SPEED           2
29                                GROCERY_ITEMS           2
28                                 GROCERY_HOUR           1
21                                      STD_ETA           1
20                                      AVG_ETA           1
0                                ORDER_PROTOCOL           1
```

In [83]:
```python
!pip install xgboost
from xgboost import XGBRegressor
from sklearn.neural_network import MLPRegressor
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting xgboost
  Downloading xgboost-3.0.0-py3-none-win_amd64.whl.metadata (2.1 kB)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages
(from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages
(from xgboost) (1.13.1)
Downloading xgboost-3.0.0-py3-none-win_amd64.whl (150.0 MB)
   ---------------------------------------- 0.0/150.0 MB ? eta -:--:--
   ---------------------------------------- 0.5/150.0 MB 2.4 MB/s eta 0:01:03
   ---------------------------------------- 1.8/150.0 MB 5.3 MB/s eta 0:00:28
    --------------------------------------- 2.4/150.0 MB 3.9 MB/s eta 0:00:38
    --------------------------------------- 3.4/150.0 MB 4.5 MB/s eta 0:00:33
   - -------------------------------------- 5.0/150.0 MB 4.9 MB/s eta 0:00:30
   - -------------------------------------- 6.6/150.0 MB 5.2 MB/s eta 0:00:28
   -- ------------------------------------- 8.1/150.0 MB 5.6 MB/s eta 0:00:26
   -- ------------------------------------- 9.7/150.0 MB 5.9 MB/s eta 0:00:24
   --- ------------------------------------ 11.3/150.0 MB 6.1 MB/s eta 0:00:23
   --- ------------------------------------ 12.6/150.0 MB 6.2 MB/s eta 0:00:23
   --- ------------------------------------ 13.9/150.0 MB 6.2 MB/s eta 0:00:22
   ---- ----------------------------------- 15.5/150.0 MB 6.3 MB/s eta 0:00:22
   ---- ----------------------------------- 17.0/150.0 MB 6.4 MB/s eta 0:00:21
   ---- ----------------------------------- 18.4/150.0 MB 6.4 MB/s eta 0:00:21
   ----- ---------------------------------- 19.9/150.0 MB 6.5 MB/s eta 0:00:21
   ----- ---------------------------------- 21.2/150.0 MB 6.4 MB/s eta 0:00:21
   ----- ---------------------------------- 22.3/150.0 MB 6.4 MB/s eta 0:00:20
   ------ --------------------------------- 23.6/150.0 MB 6.4 MB/s eta 0:00:20
   ------ --------------------------------- 24.6/150.0 MB 6.3 MB/s eta 0:00:20
   ------ --------------------------------- 25.7/150.0 MB 6.2 MB/s eta 0:00:21
   ------- -------------------------------- 27.3/150.0 MB 6.2 MB/s eta 0:00:20
   ------- -------------------------------- 28.3/150.0 MB 6.2 MB/s eta 0:00:20
   ------- -------------------------------- 29.4/150.0 MB 6.1 MB/s eta 0:00:20
   ------- -------------------------------- 29.9/150.0 MB 6.1 MB/s eta 0:00:20
   -------- ------------------------------- 30.7/150.0 MB 5.9 MB/s eta 0:00:21
   -------- ------------------------------- 31.5/150.0 MB 5.8 MB/s eta 0:00:21
   -------- ------------------------------- 32.0/150.0 MB 5.7 MB/s eta 0:00:21
   -------- ------------------------------- 33.3/150.0 MB 5.6 MB/s eta 0:00:21
   --------- ------------------------------ 34.3/150.0 MB 5.6 MB/s eta 0:00:21
   --------- ------------------------------ 35.7/150.0 MB 5.6 MB/s eta 0:00:21
   --------- ------------------------------ 37.0/150.0 MB 5.6 MB/s eta 0:00:21
   ---------- ----------------------------- 38.5/150.0 MB 5.7 MB/s eta 0:00:20
   ---------- ----------------------------- 40.4/150.0 MB 5.8 MB/s eta 0:00:20
   ----------- ---------------------------- 41.9/150.0 MB 5.8 MB/s eta 0:00:19
   ----------- ---------------------------- 43.5/150.0 MB 5.9 MB/s eta 0:00:19
   ----------- ---------------------------- 44.6/150.0 MB 5.9 MB/s eta 0:00:18
   ------------ --------------------------- 45.6/150.0 MB 5.8 MB/s eta 0:00:18
   ------------ --------------------------- 46.9/150.0 MB 5.8 MB/s eta 0:00:18
   ------------ --------------------------- 48.2/150.0 MB 5.8 MB/s eta 0:00:18
   ------------- -------------------------- 49.8/150.0 MB 5.9 MB/s eta 0:00:18
   ------------- -------------------------- 51.1/150.0 MB 5.9 MB/s eta 0:00:17
   -------------- ------------------------- 52.7/150.0 MB 5.9 MB/s eta 0:00:17
   -------------- ------------------------- 54.0/150.0 MB 5.9 MB/s eta 0:00:17
   -------------- ------------------------- 55.6/150.0 MB 5.9 MB/s eta 0:00:16
   --------------- ------------------------ 56.6/150.0 MB 5.9 MB/s eta 0:00:16
   --------------- ------------------------ 57.9/150.0 MB 5.9 MB/s eta 0:00:16
   --------------- ------------------------ 59.0/150.0 MB 5.9 MB/s eta 0:00:16
```

```
---------------- ----------------------- 60.0/150.0 MB 5.9 MB/s eta 0:00:16
---------------- ----------------------- 60.6/150.0 MB 5.8 MB/s eta 0:00:16
---------------- ----------------------- 60.8/150.0 MB 5.7 MB/s eta 0:00:16
---------------- ----------------------- 61.6/150.0 MB 5.7 MB/s eta 0:00:16
---------------- ----------------------- 62.4/150.0 MB 5.7 MB/s eta 0:00:16
---------------- ----------------------- 63.2/150.0 MB 5.6 MB/s eta 0:00:16
----------------- ---------------------- 64.0/150.0 MB 5.6 MB/s eta 0:00:16
----------------- ---------------------- 64.7/150.0 MB 5.5 MB/s eta 0:00:16
----------------- ---------------------- 65.8/150.0 MB 5.5 MB/s eta 0:00:16
----------------- ---------------------- 66.8/150.0 MB 5.5 MB/s eta 0:00:16
------------------ --------------------- 68.2/150.0 MB 5.5 MB/s eta 0:00:15
------------------ --------------------- 69.2/150.0 MB 5.5 MB/s eta 0:00:15
------------------ --------------------- 70.3/150.0 MB 5.5 MB/s eta 0:00:15
------------------- -------------------- 71.3/150.0 MB 5.5 MB/s eta 0:00:15
------------------- -------------------- 72.4/150.0 MB 5.5 MB/s eta 0:00:15
------------------- -------------------- 73.7/150.0 MB 5.5 MB/s eta 0:00:14
------------------- -------------------- 75.0/150.0 MB 5.5 MB/s eta 0:00:14
-------------------- ------------------- 76.3/150.0 MB 5.5 MB/s eta 0:00:14
-------------------- ------------------- 77.9/150.0 MB 5.5 MB/s eta 0:00:14
--------------------- ------------------ 78.9/150.0 MB 5.5 MB/s eta 0:00:13
--------------------- ------------------ 80.0/150.0 MB 5.5 MB/s eta 0:00:13
--------------------- ------------------ 80.7/150.0 MB 5.5 MB/s eta 0:00:13
--------------------- ------------------ 81.3/150.0 MB 5.5 MB/s eta 0:00:13
--------------------- ------------------ 81.5/150.0 MB 5.4 MB/s eta 0:00:13
--------------------- ------------------ 81.8/150.0 MB 5.4 MB/s eta 0:00:13
--------------------- ------------------ 82.1/150.0 MB 5.3 MB/s eta 0:00:13
--------------------- ------------------ 82.1/150.0 MB 5.3 MB/s eta 0:00:13
--------------------- ------------------ 82.3/150.0 MB 5.2 MB/s eta 0:00:14
---------------------- ----------------- 82.6/150.0 MB 5.1 MB/s eta 0:00:14
---------------------- ----------------- 83.1/150.0 MB 5.1 MB/s eta 0:00:14
---------------------- ----------------- 83.9/150.0 MB 5.0 MB/s eta 0:00:14
---------------------- ----------------- 84.7/150.0 MB 5.0 MB/s eta 0:00:14
---------------------- ----------------- 86.0/150.0 MB 5.0 MB/s eta 0:00:13
----------------------- ---------------- 87.8/150.0 MB 5.1 MB/s eta 0:00:13
----------------------- ---------------- 89.4/150.0 MB 5.1 MB/s eta 0:00:12
------------------------ --------------- 90.2/150.0 MB 5.1 MB/s eta 0:00:12
------------------------ --------------- 91.2/150.0 MB 5.1 MB/s eta 0:00:12
------------------------ --------------- 92.3/150.0 MB 5.1 MB/s eta 0:00:12
------------------------ --------------- 93.1/150.0 MB 5.0 MB/s eta 0:00:12
------------------------ --------------- 93.6/150.0 MB 5.0 MB/s eta 0:00:12
------------------------- -------------- 94.6/150.0 MB 5.0 MB/s eta 0:00:12
------------------------- -------------- 95.7/150.0 MB 5.0 MB/s eta 0:00:11
------------------------- -------------- 96.7/150.0 MB 5.0 MB/s eta 0:00:11
-------------------------- ------------- 98.0/150.0 MB 5.0 MB/s eta 0:00:11
-------------------------- ------------- 99.6/150.0 MB 5.0 MB/s eta 0:00:10
-------------------------- ------------- 100.9/150.0 MB 5.1 MB/s eta 0:00:10
--------------------------- ------------ 101.7/150.0 MB 5.0 MB/s eta 0:00:10
--------------------------- ------------ 102.5/150.0 MB 5.0 MB/s eta 0:00:10
--------------------------- ------------ 103.0/150.0 MB 5.0 MB/s eta 0:00:10
--------------------------- ------------ 103.5/150.0 MB 5.0 MB/s eta 0:00:10
--------------------------- ------------ 104.1/150.0 MB 5.0 MB/s eta 0:00:10
--------------------------- ------------ 104.6/150.0 MB 4.9 MB/s eta 0:00:10
---------------------------- ----------- 105.1/150.0 MB 4.9 MB/s eta 0:00:10
---------------------------- ----------- 105.6/150.0 MB 4.9 MB/s eta 0:00:10
---------------------------- ----------- 106.4/150.0 MB 4.9 MB/s eta 0:00:09
---------------------------- ----------- 107.5/150.0 MB 4.9 MB/s eta 0:00:09
```

```
   ----------------------------- ---------- 108.8/150.0 MB 4.9 MB/s eta 0:00:09
   ----------------------------- ---------- 110.1/150.0 MB 4.9 MB/s eta 0:00:09
   ----------------------------- ---------- 111.1/150.0 MB 4.9 MB/s eta 0:00:08
   ------------------------------ --------- 112.7/150.0 MB 4.9 MB/s eta 0:00:08
   ------------------------------ --------- 114.6/150.0 MB 4.9 MB/s eta 0:00:08
   ------------------------------- -------- 116.9/150.0 MB 5.0 MB/s eta 0:00:07
   ------------------------------- -------- 119.0/150.0 MB 5.0 MB/s eta 0:00:07
   -------------------------------- ------- 121.1/150.0 MB 5.1 MB/s eta 0:00:06
   --------------------------------- ------ 123.7/150.0 MB 5.1 MB/s eta 0:00:06
   --------------------------------- ------ 125.6/150.0 MB 5.2 MB/s eta 0:00:05
   ---------------------------------- ----- 127.7/150.0 MB 5.2 MB/s eta 0:00:05
   ---------------------------------- ----- 130.3/150.0 MB 5.3 MB/s eta 0:00:04
   ----------------------------------- ---- 132.6/150.0 MB 5.3 MB/s eta 0:00:04
   ----------------------------------- ---- 134.5/150.0 MB 5.3 MB/s eta 0:00:03
   ------------------------------------ --- 136.8/150.0 MB 5.4 MB/s eta 0:00:03
   ------------------------------------- -- 139.7/150.0 MB 5.5 MB/s eta 0:00:02
   ------------------------------------- -- 141.6/150.0 MB 5.5 MB/s eta 0:00:02
   -------------------------------------- - 143.1/150.0 MB 5.5 MB/s eta 0:00:02
   -------------------------------------- - 145.2/150.0 MB 5.5 MB/s eta 0:00:01
   ---------------------------------------  147.3/150.0 MB 5.6 MB/s eta 0:00:01
   ---------------------------------------  149.4/150.0 MB 5.6 MB/s eta 0:00:01
   ---------------------------------------  149.9/150.0 MB 5.6 MB/s eta 0:00:01
   ---------------------------------------  149.9/150.0 MB 5.6 MB/s eta 0:00:01
   ---------------------------------------  149.9/150.0 MB 5.6 MB/s eta 0:00:01
   ---------------------------------------- 150.0/150.0 MB 5.5 MB/s eta 0:00:00
Installing collected packages: xgboost
Successfully installed xgboost-3.0.0
```

In [85]:

```python
# XGBoost Regressor
xgb = XGBRegressor(random_state=42)
xgb.fit(X_train_i, y_train_i)
y_pred_xgb = xgb.predict(X_test_i)
rmse_xgb = np.sqrt(mean_squared_error(y_test_i, y_pred_xgb))
print(f"🏁 XGBoost RMSE: {rmse_xgb:.2f} seconds")
```

```
🏁 XGBoost RMSE: 11.07 seconds
```

In [87]:

```python
# MLP Regressor (Neural Net)
mlp = MLPRegressor(hidden_layer_sizes=(64, 32), max_iter=300, random_state=42)
mlp.fit(X_train_i, y_train_i)
y_pred_mlp = mlp.predict(X_test_i)
rmse_mlp = np.sqrt(mean_squared_error(y_test_i, y_pred_mlp))
print(f"🧠 MLPRegressor RMSE: {rmse_mlp:.2f} seconds")
```

```
🧠 MLPRegressor RMSE: 22.20 seconds
```

In [89]:

```python
'''
We tested three industry-standard models to predict delivery duration. LightGBM not
but also revealed the top operational levers behind late deliveries — stress index,
This gives DoorDash a reliable framework for ETA forecasting and surge preemption.
'''
```

Out[89]:

```
'\nWe tested three industry-standard models to predict delivery duration. LightGBM
not only delivered the best performance (within ±9 seconds), \nbut also revealed t
he top operational levers behind late deliveries — stress index, drive time, and g
rocery complexity. \nThis gives DoorDash a reliable framework for ETA forecasting
and surge preemption.\n'
```

In [91]:

```
df_final_ready = df_final.copy()

# Add modeled ETA bucket
df_final_ready["ETA_BUCKET"] = pd.cut(
    df_final_ready["PREDICTED_DELIVERY_DURATION"],
    bins=[0, 1200, 1800, 2700, float('inf')],
    labels=["<20m", "20-30m", "30-45m", "45m+"]
)

# Export
df_final_ready.to_csv("doordash_100k_dashboard.csv", index=False)
```

In [ ]: