UNITED AIRLINES

PEC
EXPLORE INNOVATE EXCEL

Flight Operations Difficulty Score

The ML-Optimized Engine for Proactive Ground Time Management

TEAM

HASHIRA UNITE

PROBLEM STATEMENT

Frontline teams at United Airlines are responsible for ensuring every flight departs on time and is operationally ready. However, not all flights are equally easy to manage. Certain flights pose greater complexity due to factors such as limited ground time, higher volumes of checked or carry-on baggage, and specific customer service needs that often increase with passenger load.

Currently, identifying these high-difficulty flights relies heavily on personal experience and local team knowledge. This manual approach is inconsistent, non-scalable, and risks missing opportunities for proactive resource planning across the airport.

To address this, we are tasked with developing a Flight Difficulty Score that systematically quantifies the relative complexity of each flight using the datasets provided, which span two weeks of departures from Chicago O'Hare International Airport (ORD).

DATA DESCRIPTION

Airports Data-

Airport IATA codes and ISO Country Codes

Bag+Level+Data-

Bag-Level, flight, bag tag number, issue date, and bag type

Flight Level Data-

Flight-Level operational data

PNR Remark Level Data-

PNR Remark-Level data, special service requests (SSR)

PNR+Flight+Level+Data-

PNR/Flight-Level passenger counts

OBJECTIVE

The goal is to design a data-driven framework that:

- Calculates a Flight Difficulty Score for each flight using flight-level, customer, and station-level data.
- Identifies the primary operational drivers contributing to flight difficulty to enable proactive planning and optimized resource allocation.

DELIVERABLES

EXPLORATORY DATA ANALYSIS (EDA)

FLIGHT DIFFICULTY SCORE DEVELOPMENT

POST-ANALYSIS & OPERATIONAL INSIGHTS

EXPLORATORY DATA ANALYSIS (EDA)

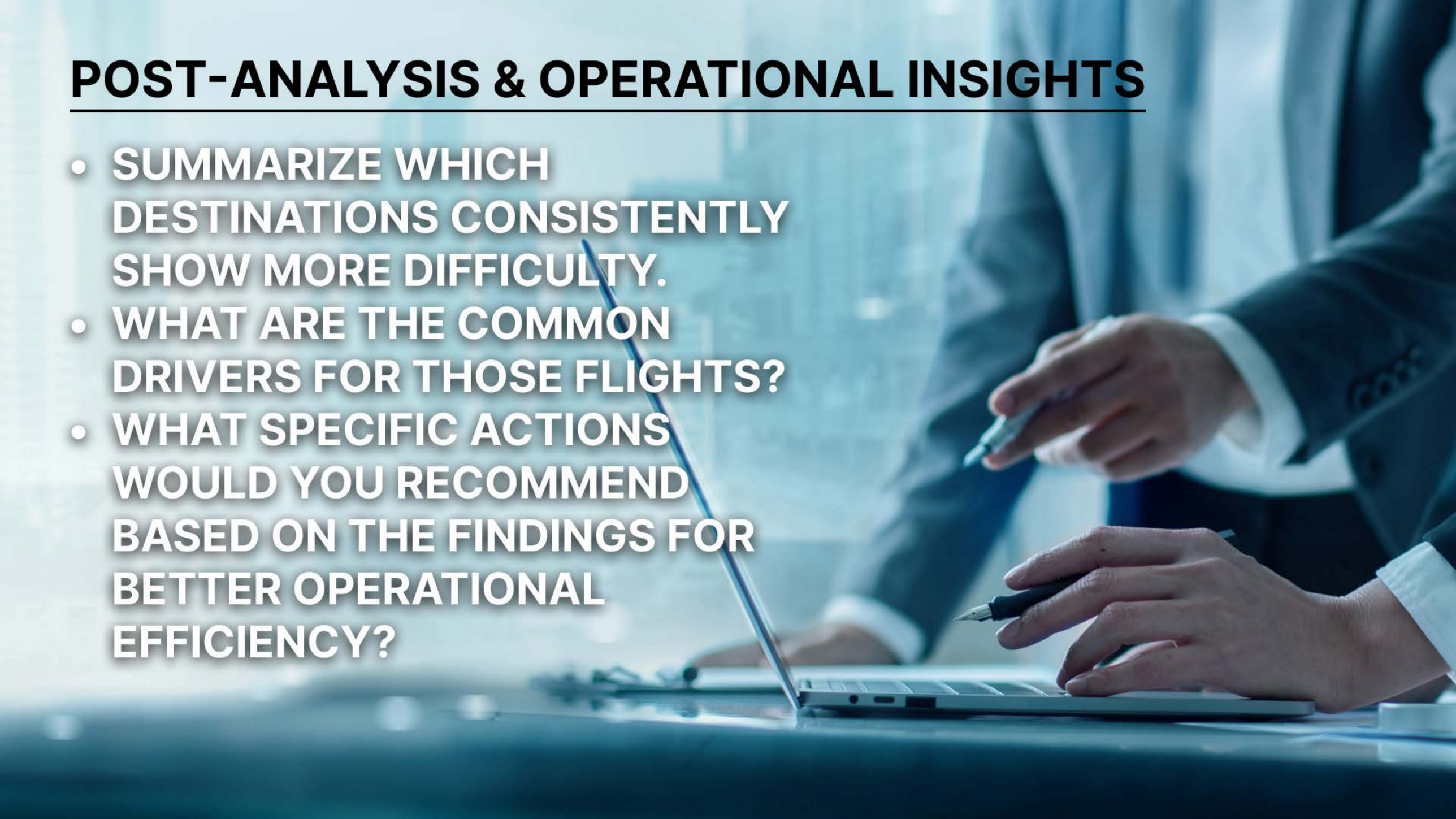
- WHAT IS THE AVERAGE DELAY AND WHAT PERCENTAGE OF FLIGHTS DEPART LATER THAN SCHEDULED?
- HOW MANY FLIGHTS HAVE SCHEDULED GROUND TIME CLOSE TO OR BELOW THE MINIMUM TURN MINS?
- WHAT IS THE AVERAGE RATIO OF TRANSFER BAGS VS. CHECKED BAGS ACROSS FLIGHTS?
- HOW DO PASSENGER LOADS COMPARE ACROSS FLIGHTS, AND DO HIGHER LOADS CORRELATE WITH OPERATIONAL DIFFICULTY?
- ARE HIGH SPECIAL SERVICE REQUESTS FLIGHTS ALSO HIGH-DELAY AFTER CONTROLLING FOR LOAD?

FLIGHT DIFFICULTY SCORE DEVELOPMENT

- BUILD A SYSTEMATIC DAILY-LEVEL SCORING APPROACH THAT RESETS EVERY DAY. BELOW ARE THE OUTPUTS REQUIRED:
  - RANKING: WITHIN EACH DAY, ORDER FLIGHTS BY THEIR DIFFICULTY SCORE SO THAT THE HIGHEST-RANKED FLIGHTS REPRESENT THE MOST DIFFICULT TO MANAGE.
  - CLASSIFICATION: GROUP FLIGHTS INTO THREE CATEGORIES — DIFFICULT, MEDIUM, OR EASY — BASED ON RANK DISTRIBUTION.

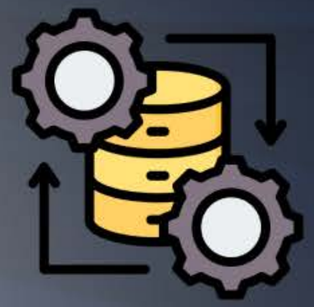
POST-ANALYSIS & OPERATIONAL INSIGHTS

- SUMMARIZE WHICH DESTINATIONS CONSISTENTLY SHOW MORE DIFFICULTY.
- WHAT ARE THE COMMON DRIVERS FOR THOSE FLIGHTS?
- WHAT SPECIFIC ACTIONS WOULD YOU RECOMMEND BASED ON THE FINDINGS FOR BETTER OPERATIONAL EFFICIENCY?



PROJECT

PIPELINE



DATA PREPROCESSING

EXPLORATORY DATA ANALYSIS (EDA)

MACHINE LEARNING &
MODEL TRAINING

INSIGHTS ANALYSIS & ACTION PLANNING

DATA PREPROCESSING

Importing Libraries

```
mporting the pandas library – a powerful Python library used for data manipulation and analysis.
ort pandas as pd

mporting the numpy library – used for numerical and mathematical operations on large, multi-dimensional arrays and matrices.
ort numpy as np

mporting matplotlib's pyplot module – used for creating visualizations such as line charts, bar plots, histograms, scatter plots, e
ort matplotlib.pyplot as plt

mporting seaborn – a statistical data visualization library built on top of matplotlib.
ort seaborn as sns
```

Accessing The CSVs

```
#Accessing the CSVs
airport_data=pd.read_csv('/content/Airports Data.csv')
bag_level_data=pd.read_csv('/content/Bag+Level+Data.csv')
flight_level_data=pd.read_csv('/content/Flight Level Data.csv')
pnr_remark_level_data=pd.read_csv('/content/PNR Remark Level Data.csv')
pnr_flight_level_data=pd.read_csv('/content/PNR+Flight+Level+Data.csv')
```

SAFELY LOADING & CLEANING DISPARATE DATA AND FINALLY MERGING THE AGGREGATE ,MAKING IT READY FOR THE SUBSEQUENT MACHINE LEARNING-BASED DIFFICULTY SCORING.

```python
# --- Aggregations ---

# Count total, transfer, and checked bags per flight
if not bags.empty:
    bags_agg = bags.groupby(
        ["flight_number", "scheduled_departure_date_local"]
    ).agg(
        total_bags=("bag_type", "count"),
        transfer_bags=("bag_type", lambda x: (x == "Transfer").sum()),
        checked_bags=("bag_type", lambda x: (x == "Checked").sum())
    ).reset_index()
else:
    bags_agg = pd.DataFrame(columns=["flight_number","scheduled_departure_date_local","total_bags","transfer_bags","checked_bags"])
```

```python
# Aggregate passenger info per flight
if not pnr_flight.empty:
    pnr_agg = pnr_flight.groupby(
        ["flight_number", "scheduled_departure_date_local"]
    ).agg(
        total_pax=("total_pax", "sum"),
        children=("is_child", "sum"),
        stroller_users=("is_stroller_user", "sum"),
        lap_children=("lap_child_count", "sum"),
        basic_economy=("basic_economy_pax", "sum")
    ).reset_index()
else:
    pnr_agg = pd.DataFrame(columns=["flight_number","scheduled_departure_date_local","total_pax","children","stroller_users","lap_children","basic_economy"])
```

```python
# Count total special service requests per flight
if not pnr_remarks.empty:
    remarks_agg = pnr_remarks.groupby(
        ["flight_number"]
    ).agg(
        special_service_count=("special_service_request", "count")
    ).reset_index()
else:
    remarks_agg = pd.DataFrame(columns=["flight_number","special_service_count"])
```

```python
# --- Merge all aggregated data into one master DataFrame ---
merged = flights.copy()

# Merge bag, PNR, and remark aggregates via left joins to keep all flights
if not bags_agg.empty:
    merged = merged.merge(bags_agg, on=["flight_number","scheduled_departure_date_local"], how="left")

if not pnr_agg.empty:
    merged = merged.merge(pnr_agg, on=["flight_number","scheduled_departure_date_local"], how="left")

if not remarks_agg.empty:
    merged = merged.merge(remarks_agg, on="flight_number", how="left")

# Add departure and arrival airport country codes
if not airports.empty:
    merged = merged.merge(
        airports.add_prefix("dep_"),
        left_on="scheduled_departure_station_code",
        right_on="dep_airport_iata_code",
        how="left"
    ).merge(
        airports.add_prefix("arr_"),
        left_on="scheduled_arrival_station_code",
        right_on="arr_airport_iata_code",
        how="left"
    )
```


```python
# Add departure and arrival airport country codes
if not airports.empty:
    merged = merged.merge(
        airports.add_prefix("dep_"),
        left_on="scheduled_departure_station_code",
        right_on="dep_airport_iata_code",
        how="left"
    ).merge(
        airports.add_prefix("arr_"),
        left_on="scheduled_arrival_station_code",
        right_on="arr_airport_iata_code",
        how="left"
    )
```

MAKING IT AN AGGREGATED DATASET

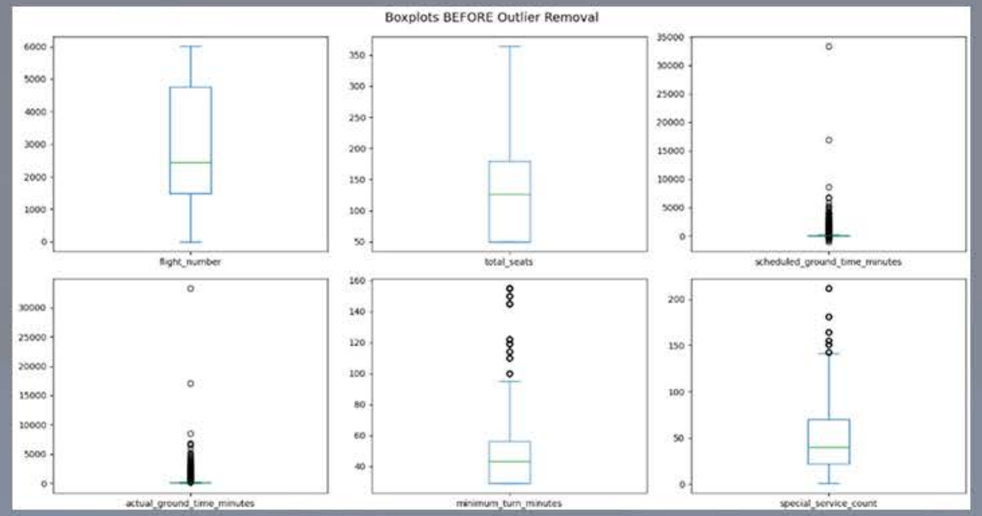
Final Master Dataset

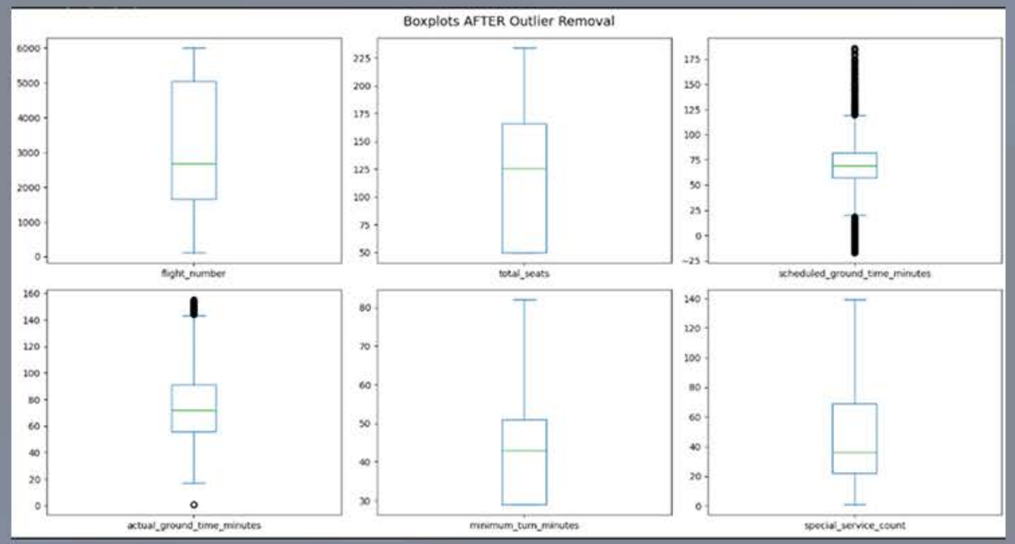
```
final_master_dataset=pd.read_csv('/content/final_master_dataset.csv')
final_master_dataset.head(5)
```

	company_id	flight_number	scheduled_departure_date_local	scheduled_departure_station_code	scheduled_arrival_station_code	scheduled_departure_datetime_local
0	OO	4792	2025-08-04	ORD	ROA	2025-08-04T17:57:00Z
1	UA	920	2025-08-03	ORD	LHR	2025-08-03T18:05:00Z
2	UA	1776	2025-08-10	ORD	PHL	2025-08-10T18:20:00Z
3	OO	5790	2025-08-06	ORD	CRW	2025-08-06T18:20:00Z
4	UA	1398	2025-08-05	ORD	ATL	2025-08-05T18:20:00Z

OUTLIER DETECTION AND REMOVAL

```python
# Function to remove outliers using IQR
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
```





BEFORE

```
Original shape: (8099, 20)
Cleaned shape: (6053, 20)
```

AFTER

DATA TRANSFORMATION AND IMPUTATION

SPLITTED ALL COMBINED ISO DATETIME COLUMNS INTO SEPARATE DATE AND TIME FIELDS AND SYSTEMATICALLY FILLED MISSING DATE VALUES USING FORWARD/ BACKWARD FILL AND MISSING TIME VALUES USING THE MODE

```python
# Helper function to split ISO datetime string
def split_datetime(value):
    if pd.isna(value) or value == '':
        return pd.NA, pd.NA
    parts = value.split('T')
    date_part = parts[0]
    time_part = parts[1].replace('Z', '') if len(parts) > 1 else pd.NA
    return date_part, time_part

# Create new columns
new_df['Schedule_Departure_Date'], new_df['Schedule_Departure_Time'] = zip(*new_df['scheduled_departure_datetime_local'].apply(split_datetime))
new_df['Schedule_Arrival_Date'], new_df['Schedule_Arrival_Time'] = zip(*new_df['scheduled_arrival_datetime_local'].apply(split_datetime))
new_df['Actual_Departure_Date'], new_df['Actual_Departure_Time'] = zip(*new_df['actual_departure_datetime_local'].apply(split_datetime))
new_df['Actual_Arrival_Date'], new_df['Actual_Arrival_Time'] = zip(*new_df['actual_arrival_datetime_local'].apply(split_datetime))

# Drop the original datetime columns
new_df.drop(
    columns=[
        'scheduled_departure_datetime_local',
        'scheduled_arrival_datetime_local',
        'actual_departure_datetime_local',
        'actual_arrival_datetime_local',
        'scheduled_departure_date_local'
    ],
    inplace=True
)
```

HANDLING MISSING VALUES & REMOVING DUPLICATES

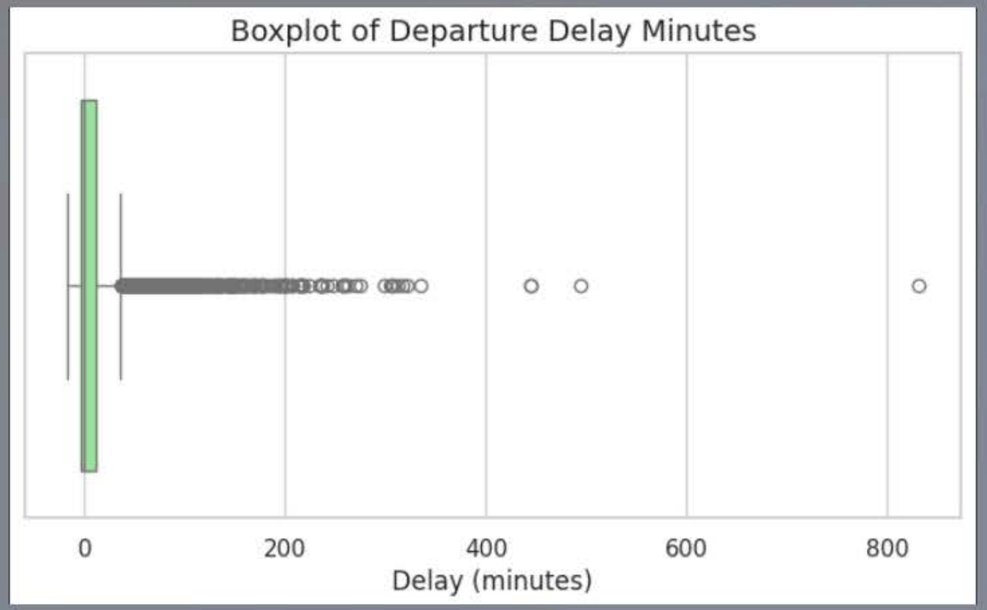
```python
# Handle missing values meaningfully

# Fill missing date values with forward fill then backward fill
date_cols = [
    'Schedule_Departure_Date',
    'Schedule_Arrival_Date',
    'Actual_Departure_Date',
    'Actual_Arrival_Date'
]
new_df[date_cols] = new_df[date_cols].ffill().bfill()

# Fill missing time values with the mode of each column
time_cols = [
    'Schedule_Departure_Time',
    'Schedule_Arrival_Time',
    'Actual_Departure_Time',
    'Actual_Arrival_Time'
]
for col in time_cols:
    mode_time = new_df[col].mode()[0] if not new_df[col].mode().empty else '00:00:00'
    new_df[col].fillna(mode_time, inplace=True)

# Remove duplicate rows
new_df.drop_duplicates(inplace=True)
```

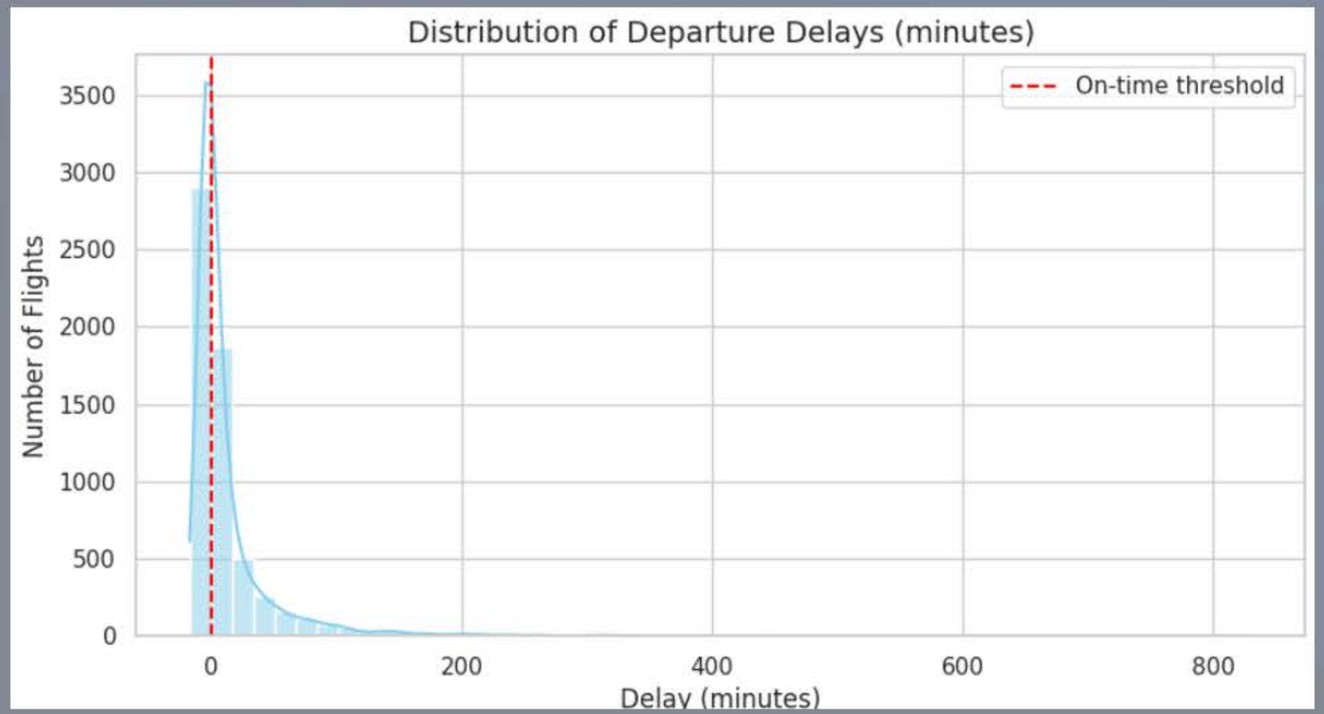
EXPLORATORY DATA ANALYSIS (EDA)

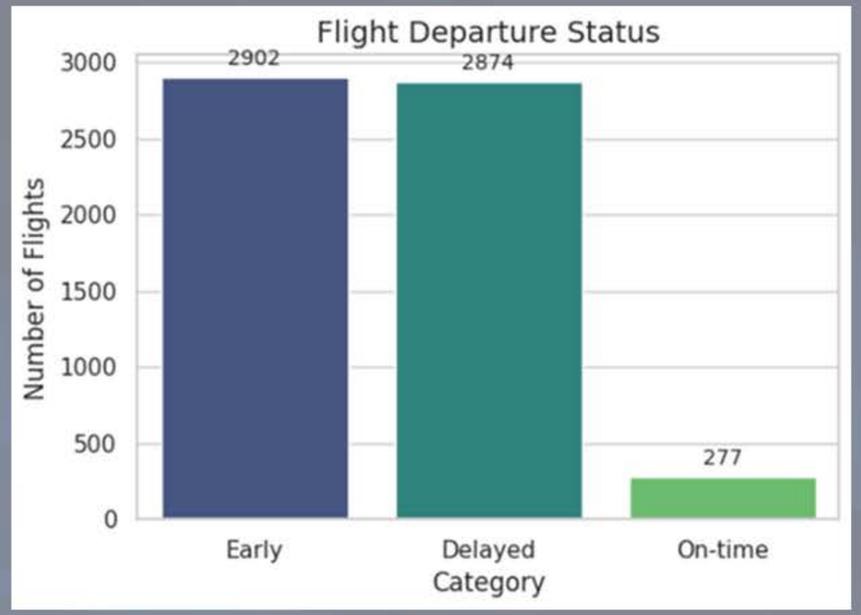
AVERAGE DELAY & PERCENTAGE OF FLIGHTS DEPARTING LATER THAN SCHEDULED

```python
# Step 1: Convert date columns to datetime
df_latest['Schedule_Departure_Date'] = pd.to_datetime(df_latest['Schedule_Departure_Date'], errors='coerce')
df_latest['Actual_Departure_Date'] = pd.to_datetime(df_latest['Actual_Departure_Date'], errors='coerce')

# Step 2: Convert time columns to string
df_latest['Schedule_Departure_Time'] = df_latest['Schedule_Departure_Time'].astype(str).str.replace('Z', '', regex=False)
df_latest['Actual_Departure_Time'] = df_latest['Actual_Departure_Time'].astype(str).str.replace('Z', '', regex=False)

# Step 3: Combine date and time into datetime
df_latest['scheduled_departure_datetime'] = pd.to_datetime(
    df_latest['Schedule_Departure_Date'].dt.strftime('%Y-%m-%d') + ' ' + df_latest['Schedule_Departure_Time'],
    errors='coerce',
    utc=False
)
df_latest['actual_departure_datetime'] = pd.to_datetime(
    df_latest['Actual_Departure_Date'].dt.strftime('%Y-%m-%d') + ' ' + df_latest['Actual_Departure_Time'],
    errors='coerce',
    utc=False
)

# Step 4: Remove timezone if exists (make both tz-naive)
df_latest['scheduled_departure_datetime'] = df_latest['scheduled_departure_datetime'].dt.tz_localize(None)
df_latest['actual_departure_datetime'] = df_latest['actual_departure_datetime'].dt.tz_localize(None)
```

```python
# Step 5: Calculate delay in minutes
df_latest['departure_delay_minutes'] = (
    df_latest['actual_departure_datetime'] - df_latest['scheduled_departure_datetime']
).dt.total_seconds() / 60

# Step 6: Compute KPIs
average_delay = df_latest['departure_delay_minutes'].mean()
delayed_flights = (df_latest['departure_delay_minutes'] > 0).sum()
total_flights = len(df_latest)
percentage_delayed = (delayed_flights / total_flights) * 100

print(f"Average Departure Delay: {average_delay:.2f} minutes")
print(f"Percentage of Flights Delayed: {percentage_delayed:.2f}%")
print(f"Delayed Flights: {delayed_flights} out of {total_flights}")

print(f"\nDelay Statistics:")
print(f"Minimum Delay: {df_latest['departure_delay_minutes'].min():.2f} minutes")
print(f"Maximum Delay: {df_latest['departure_delay_minutes'].max():.2f} minutes")
print(f"Median Delay: {df_latest['departure_delay_minutes'].median():.2f} minutes")
```







Result Summary:

Average Departure Delay: 12.28 minutes

Percentage of Flights Delayed: 47.48%

Delayed Flights: 2874 out of 6053

Delay Statistics: Minimum Delay: -17.00 minutes

Maximum Delay: 831.00 minutes

Median Delay: 0.00 minutes

HOW MANY FLIGHTS HAVE SCHEDULED GROUND TIME CLOSE TO OR BELOW THE MINIMUM TURN MINS?

```
threshold = 5  # Allowable buffer (in minutes) for ground time
# Filter flights where ground time is close to or below minimum turn time
close_or_below_flights = df_latest[df_latest['scheduled_ground_time_minutes'] <= (df_latest['minimum_turn_minutes'] + threshold)]

count_close_or_below = len(close_or_below_flights)  # Count such flights

# Output results
print(f"Number of flights with scheduled ground time close to or below minimum turn minutes: {count_close_or_below}")
print(f"Total flights in dataset: {len(df_latest)}")
print(f"Percentage of flights with tight scheduling: {count_close_or_below/len(df_latest)*100:.2f}%")
```

Result Summary:

Number of flights with scheduled ground time close to or below minimum turn minutes: 480

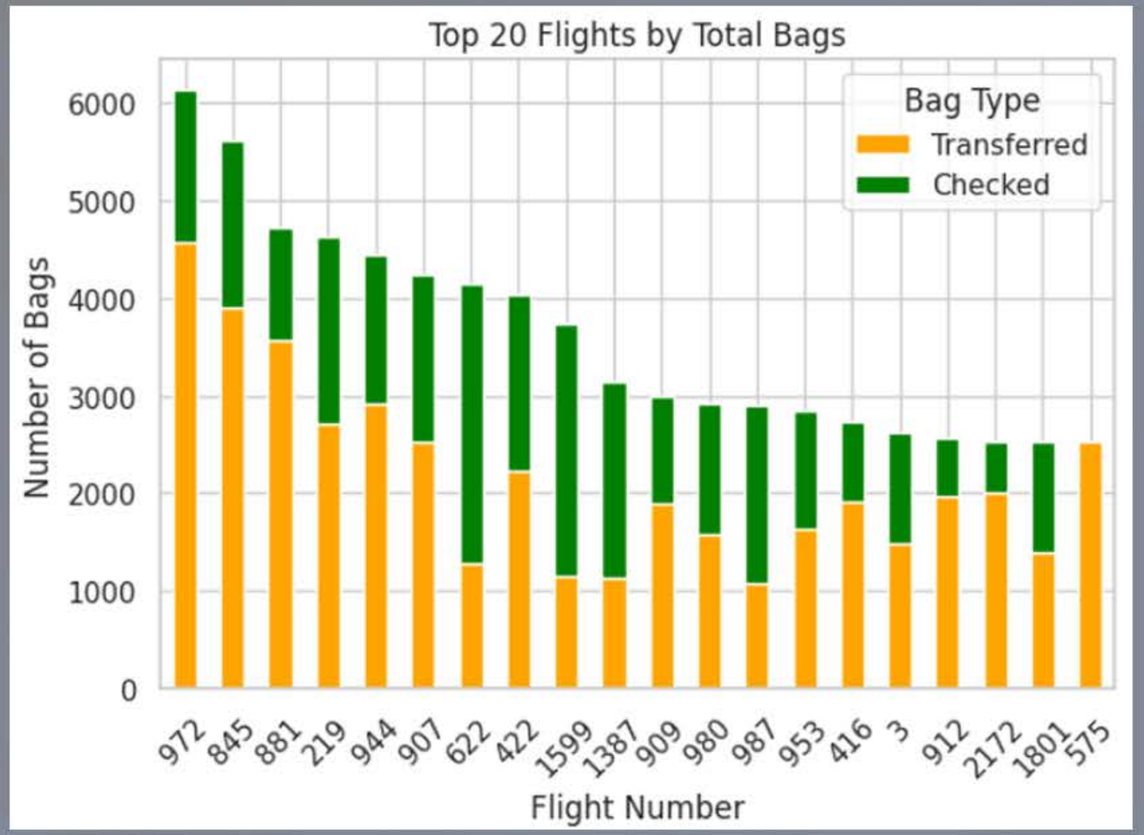
Total flights in dataset: 6053

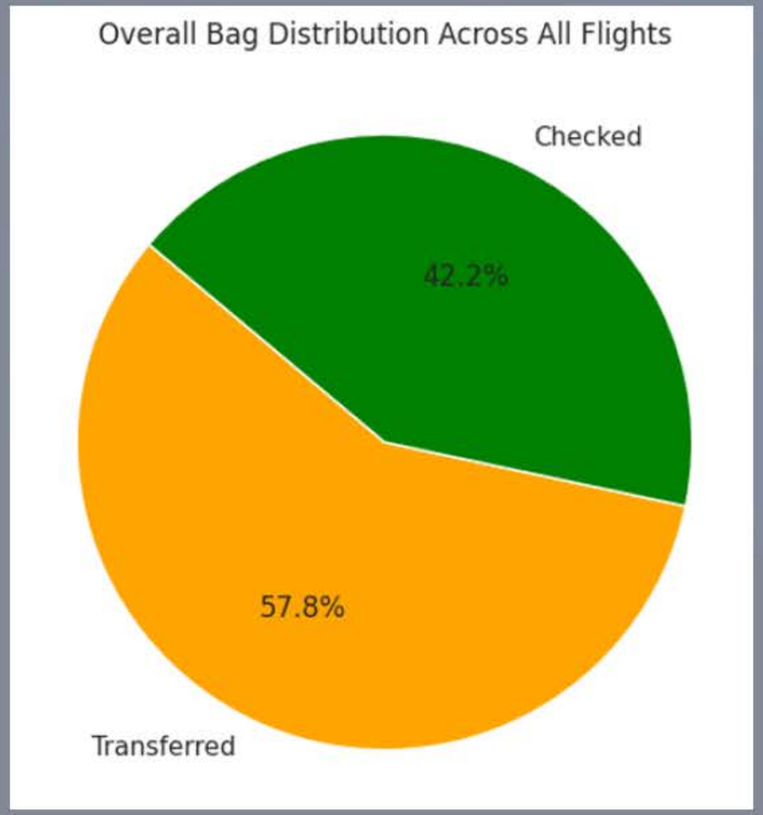
Percentage of flights with tight scheduling: 7.93%

WHAT IS THE AVERAGE RATIO OF TRANSFER BAGS VS. CHECKED BAGS ACROSS FLIGHTS?

```python
# Encode bag types
def encode_bag_type(value):
    if pd.isna(value):
        return pd.NA
    value_lower = value.lower()
    if 'transfer' in value_lower:
        return 'Transferred'
    elif 'origin' in value_lower:
        return 'Checked'
    else:
        return pd.NA

bag_level_data['bag_type_encoded'] = bag_level_data['bag_type'].apply(encode_bag_type)

# Count Transferred and Checked per flight
bag_counts = bag_level_data.groupby('flight_number')['bag_type_encoded'].value_counts().unstack(fill_value=0)

# Remove flights with 0 Checked AND 0 Transferred
bag_counts = bag_counts[(bag_counts.get('Transferred', 0) > 0) | (bag_counts.get('Checked', 0) > 0)]

# Compute ratio safely (only divide by Checked > 0)
bag_counts['transfer_to_checked_ratio'] = bag_counts.apply(
    lambda row: row['Transferred'] / row['Checked'] if row['Checked'] > 0 else row['Transferred'], axis=1
)
```

```python
# Compute average and median ratios
average_ratio = bag_counts['transfer_to_checked_ratio'].mean()
median_ratio = bag_counts['transfer_to_checked_ratio'].median()

# Output
print(f"Average Transfer-to-Checked Ratio: {average_ratio:.2f}")
print(f"Median Transfer-to-Checked Ratio: {median_ratio:.2f}")
```





Result Summary:

Average Transfer-to-Checked Ratio: 18.66

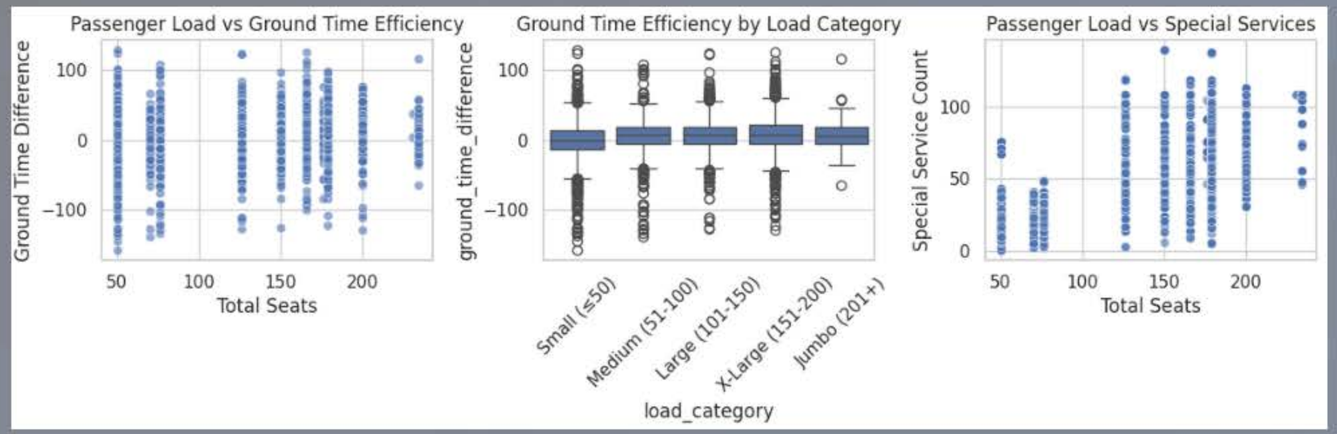
Median Transfer-to-Checked Ratio: 2.00

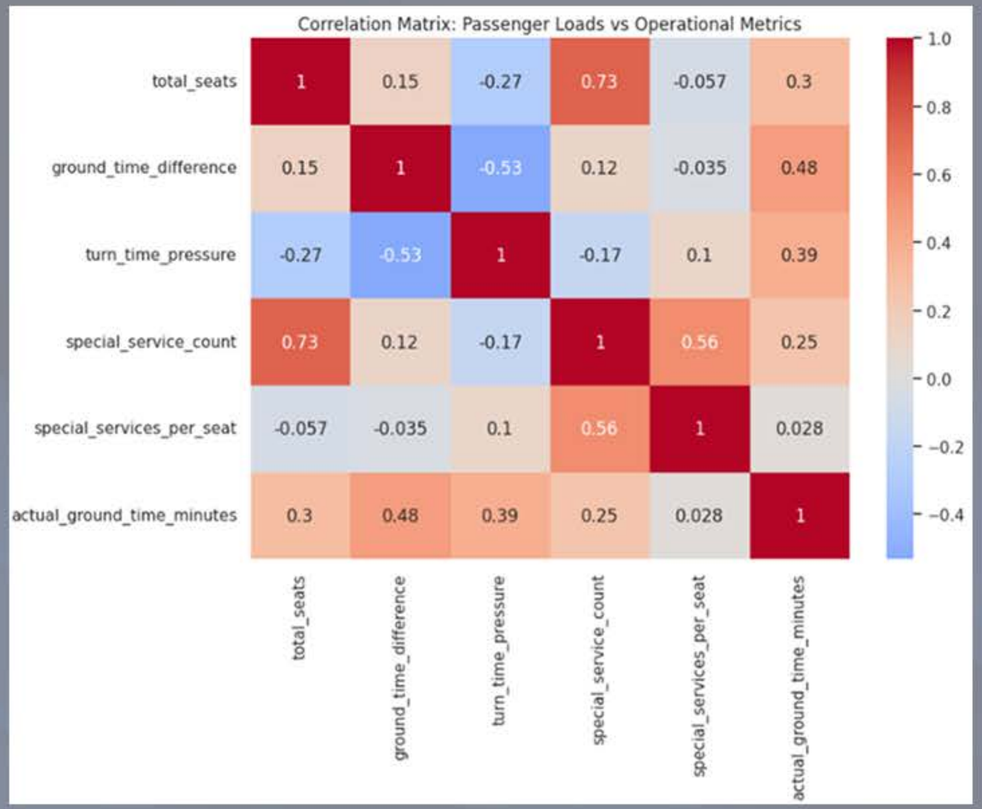
HOW DO PASSENGER LOADS COMPARE ACROSS FLIGHTS, AND DO HIGHER LOADS CORRELATE WITH OPERATIONAL DIFFICULTY?

```python
# Analyze passenger loads and operational difficulty
# 1. Basic statistics of passenger loads
print("=== PASSENGER LOAD ANALYSIS ===")
print(f"Total seats statistics:")
print(f"Min: {df_latest['total_seats'].min()}")
print(f"Max: {df_latest['total_seats'].max()}")
print(f"Mean: {df_latest['total_seats'].mean():.2f}")
print(f"Median: {df_latest['total_seats'].median()}")
print(f"Standard deviation: {df_latest['total_seats'].std():.2f}")

# 2. Distribution of passenger loads by carrier type
print(f"\n=== PASSENGER LOADS BY CARRIER TYPE ===")
carrier_loads = df_latest.groupby('carrier')['total_seats'].agg(['mean', 'median', 'min', 'max', 'count'])
print(carrier_loads)

# 3. Define operational difficulty metrics
# We'll use ground time efficiency and delays as proxies for operational difficulty

# Ground time efficiency: actual vs scheduled ground time
df_latest['ground_time_difference'] = df_latest['actual_ground_time_minutes'] - df_latest['scheduled_ground_time_minutes']
df_latest['ground_time_efficiency'] = df_latest['scheduled_ground_time_minutes'] / df_latest['actual_ground_time_minutes']

# Turn time pressure: scheduled ground time vs minimum required
df_latest['turn_time_pressure'] = df_latest['scheduled_ground_time_minutes'] - df_latest['minimum_turn_minutes']
```

```python
# Special services as complexity indicator
df_latest['special_services_per_seat'] = df_latest['special_service_count'] / df_latest['total_seats']

print(f"\n=== OPERATIONAL DIFFICULTY METRICS ===")
print(f"Ground time difference stats (actual - scheduled):")
print(f"  Mean: {df_latest['ground_time_difference'].mean():.2f} minutes")
print(f"  Median: {df_latest['ground_time_difference'].median():.2f} minutes")

print(f"\nTurn time pressure stats (scheduled - minimum):")
print(f"  Mean: {df_latest['turn_time_pressure'].mean():.2f} minutes")
print(f"  Median: {df_latest['turn_time_pressure'].median():.2f} minutes")

# 4. Correlation analysis between passenger loads and operational difficulty
print(f"\n=== CORRELATION ANALYSIS ===")
correlation_metrics = ['total_seats', 'ground_time_difference', 'turn_time_pressure',
                       'special_service_count', 'special_services_per_seat', 'actual_ground_time_minutes']

correlation_matrix = df_latest[correlation_metrics].corr()

print("Correlation matrix:")
print(correlation_matrix.round(3))

# Specific correlations with passenger loads
print(f"\nCorrelation of total_seats with operational metrics:")
for metric in ['ground_time_difference', 'turn_time_pressure', 'special_service_count', 'actual_ground_time_minutes']:
    corr = df_latest['total_seats'].corr(df_latest[metric])
    print(f"  {metric}: {corr:.3f}")
```

```python
# 5. Analyze by load categories
print(f"\n=== ANALYSIS BY PASSENGER LOAD CATEGORIES ===")
# Create load categories
df_latest['load_category'] = pd.cut(df_latest['total_seats'],
                                    bins=[0, 50, 100, 150, 200, 300],
                                    labels=['Small (≤50)', 'Medium (51-100)', 'Large (101-150)',
                                            'X-Large (151-200)', 'Jumbo (201+)'])

load_category_stats = df_latest.groupby('load_category').agg({
    'total_seats': 'count',
    'ground_time_difference': 'mean',
    'turn_time_pressure': 'mean',
    'special_service_count': 'mean',
    'actual_ground_time_minutes': 'mean'
}).round(2)

load_category_stats.columns = ['flight_count', 'avg_ground_time_diff', 'avg_turn_pressure',
                               'avg_special_services', 'avg_actual_ground_time']
print(load_category_stats)

# 6. Fleet type analysis
print(f"\n=== FLEET TYPE ANALYSIS ===")
fleet_stats = df_latest.groupby('fleet_type').agg({
    'total_seats': ['mean', 'median', 'count'],
    'ground_time_difference': 'mean',
    'turn_time_pressure': 'mean'
}).round(2)

print(fleet_stats)
```





Result Summary:

Load Distribution: There's a sharp distinction between Express flights (≈60 seats) and Mainline flights (≈163 seats), with Mainline aircraft accounting for roughly half the flight count (3,065).

Load Correlation with Difficulty: Flights with higher seat counts (Load Categories Large, X-Large, and Jumbo) show a strong positive correlation with key difficulty factors:

Special Service Count: Very strong positive correlation (0.733 with total_seats), meaning larger planes carry significantly more resource-intensive requests.

Ground Time Difference (Delay): Positive correlation (0.149) and a clear escalating trend in the categorical analysis: the average delay (avg_ground_time_diff) worsens consistently from -2.03 minutes (Small) to +8.54 minutes (X-Large).

Turn Time Pressure: A negative correlation (-0.275 with total_seats) suggests larger aircraft have less cushion (lower turn_time_pressure, e.g., only 18.59 mins for X-Large vs. 38.94 mins for Small), making them more sensitive to delays.

ARE HIGH SPECIAL SERVICE REQUESTS FLIGHTS ALSO HIGH-DELAY AFTER CONTROLLING FOR LOAD?

```python
# Analyze if high special service flights have more delays after controlling for load

print("=== SPECIAL SERVICE REQUESTS vs DELAYS ANALYSIS ===")

# 1. Calculate delay metrics
# Using ground time difference as our primary delay metric
df_latest['ground_time_delay'] = df_latest['actual_ground_time_minutes'] - df_latest['scheduled_ground_time_minutes']

# Also calculate departure delay if we have the data
# Check if we have actual vs scheduled departure times to calculate departure delays
try:
    # Convert time strings to datetime if needed
    if 'Actual_Departure_Time' in df_latest.columns and 'Schedule_Departure_Time' in df_latest.columns:
        # This would require proper datetime conversion - simplified approach
        print("Departure time data available for delay calculation")
except:
    print("Using ground time difference as primary delay metric")

print(f"\nDelay Statistics:")
print(f"Ground time delay - Mean: {df_latest['ground_time_delay'].mean():.2f} min")
print(f"Ground time delay - Median: {df_latest['ground_time_delay'].median():.2f} min")
print(f"Ground time delay - Std: {df_latest['ground_time_delay'].std():.2f} min")

print(f"\nSpecial Service Statistics:")
print(f"Special service count - Mean: {df_latest['special_service_count'].mean():.2f}")
print(f"Special service count - Median: {df_latest['special_service_count'].median()}")
print(f"Special service count - Std: {df_latest['special_service_count'].std():.2f}")

# 2. Categorize flights by special service levels
df_latest['special_service_level'] = pd.qcut(df_latest['special_service_count'],
                                             q=4,
                                             labels=['Low', 'Medium-Low', 'Medium-High', 'High'])
```

```python
# 3. Basic correlation analysis
print(f"\n=== BASIC CORRELATIONS ===")
corr_special_delay = df_latest['special_service_count'].corr(df_latest['ground_time_delay'])
corr_load_delay = df_latest['total_seats'].corr(df_latest['ground_time_delay'])
corr_special_load = df_latest['special_service_count'].corr(df_latest['total_seats'])

print(f"Correlation between special services and delay: {corr_special_delay:.3f}")
print(f"Correlation between passenger load and delay: {corr_load_delay:.3f}")
print(f"Correlation between special services and passenger load: {corr_special_load:.3f}")

# 4. Group analysis by special service levels
print(f"\n=== DELAYS BY SPECIAL SERVICE LEVELS ===")
service_level_stats = df_latest.groupby('special_service_level').agg({
    'ground_time_delay': ['mean', 'median', 'count'],
    'total_seats': 'mean',
    'special_service_count': 'mean'
}).round(2)

print(service_level_stats)

# 5. Statistical test - ANOVA to check if delay means differ across service levels
print(f"\n=== STATISTICAL TEST - ANOVA ===")
low_delay = df_latest[df_latest['special_service_level'] == 'Low']['ground_time_delay']
medium_low_delay = df_latest[df_latest['special_service_level'] == 'Medium-Low']['ground_time_delay']
medium_high_delay = df_latest[df_latest['special_service_level'] == 'Medium-High']['ground_time_delay']
high_delay = df_latest[df_latest['special_service_level'] == 'High']['ground_time_delay']

f_stat, p_value = stats.f_oneway(low_delay, medium_low_delay, medium_high_delay, high_delay)
print(f"ANOVA F-statistic: {f_stat:.3f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("→ Significant difference in delays across special service levels")
else:
    print("→ No significant difference in delays across special service levels")
```

```python
# 6. Regression analysis - controlling for passenger load
print(f"\n=== REGRESSION ANALYSIS (CONTROLLING FOR LOAD) ===")

# Prepare data for regression
X = df_latest[['special_service_count', 'total_seats']]
X = sm.add_constant(X)  # Add constant term
y = df_latest['ground_time_delay']

# Remove any rows with missing values
valid_mask = ~(X.isna().any(axis=1) | y.isna())
X_clean = X[valid_mask]
y_clean = y[valid_mask]

# Fit regression model
model = sm.OLS(y_clean, X_clean).fit()

print(model.summary())

# 7. Partial correlation analysis
print(f"\n=== PARTIAL CORRELATION ANALYSIS ===")
# Partial correlation between special services and delay, controlling for load
def partial_corr(x, y, control):
    """Calculate partial correlation between x and y, controlling for control variable"""
    # Residuals of x after controlling for control
    model_x = sm.OLS(x, sm.add_constant(control)).fit()
    residuals_x = model_x.resid

    # Residuals of y after controlling for control
    model_y = sm.OLS(y, sm.add_constant(control)).fit()
    residuals_y = model_y.resid

    # Correlation of residuals
    return np.corrcoef(residuals_x, residuals_y)[0, 1]

# Calculate partial correlation
partial_corr_value = partial_corr(
    df_latest['special_service_count'],
    df_latest['ground_time_delay'],
    df_latest['total_seats']
)
```

```python
# 8. Analysis by load categories to control for load effects
print(f"\n=== ANALYSIS WITHIN LOAD CATEGORIES ===")

# Create load categories
df_latest['load_category'] = pd.cut(df_latest['total_seats'],
                                    bins=[0, 50, 100, 150, 200, 300],
                                    labels=['Small (≤50)', 'Medium (51-100)', 'Large (101-150)',
                                            'X-Large (151-200)', 'Jumbo (201+)'])

# Analyze correlation within each load category
for category in df_latest['load_category'].cat.categories:
    category_data = df_latest[df_latest['load_category'] == category]
    if len(category_data) > 10:  # Only calculate if sufficient data
        corr_in_category = category_data['special_service_count'].corr(category_data['ground_time_delay'])
        print(f"{category}: Correlation = {corr_in_category:.3f} (n={len(category_data)})")

# 9. Extreme cases analysis
print(f"\n=== EXTREME CASES ANALYSIS ===")
# Flights with very high special services but normal loads
high_services = df_latest[df_latest['special_service_count'] > df_latest['special_service_count'].quantile(0.75)]
normal_services = df_latest[df_latest['special_service_count'] <= df_latest['special_service_count'].quantile(0.75)]

# Compare delays while controlling for similar load ranges
load_range = (df_latest['total_seats'].quantile(0.25), df_latest['total_seats'].quantile(0.75))
comparable_flights = df_latest[
    (df_latest['total_seats'] >= load_range[0]) &
    (df_latest['total_seats'] <= load_range[1])
]

high_services_comparable = comparable_flights[comparable_flights['special_service_count'] > comparable_flights['special_service_count'].median()]
normal_services_comparable = comparable_flights[comparable_flights['special_service_count'] <= comparable_flights['special_service_count'].median()]

print(f"Comparable load range: {load_range[0]:.0f} - {load_range[1]:.0f} seats")
print(f"High services group delay: {high_services_comparable['ground_time_delay'].mean():.2f} min (n={len(high_services_comparable)})")
print(f"Normal services group delay: {normal_services_comparable['ground_time_delay'].mean():.2f} min (n={len(normal_services_comparable)})")
```

Result Summary:

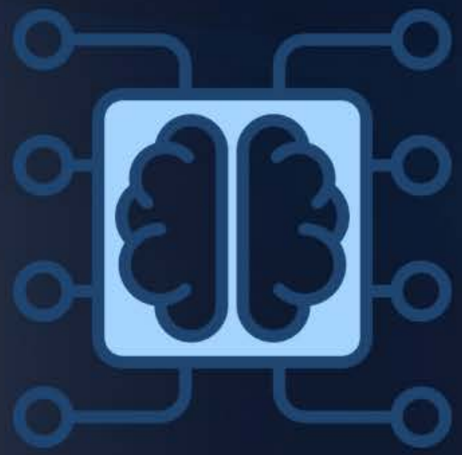
High special service requests do not directly drive increased delay once passenger load is accounted for; the observed relationship is largely due to *confounding by aircraft size*.

Initial Observation (Spurious Link): Flights with High service levels saw an average delay of 8.35 minutes, compared to only 0.92 minutes for Low service flights, showing a strong difference (ANOVA P-value 0.000).

Controlling for Load (The Critical Finding): When passenger load (total_seats) was included in the regression model to control for its effect: The special_service_count coefficient became statistically insignificant (P>0.05 or 0.484), meaning special requests do not independently cause delay.

The total_seats coefficient remained highly significant (P=0.000), confirming that load is the true, independent driver of operational difficulty.

Confirmation: The Partial Correlation between services and delay, after removing the influence of load, dropped almost to zero (0.009), confirming that services are a symptom of high-load aircraft, not an independent cause of delay.



MACHINE LEARNING & MODEL TRAINING

FLIGHT DIFFICULTY SCORE (FDS) DEVELOPMENT

-AGGREGATED AND NORMALIZED FIVE KEY OPERATIONAL KPIS (DELAY, GROUND TIME INVERSE, BAGS, TRANSFER RATIO, SERVICES) AT THE FLIGHT LEVEL.
-TRAINED A RANDOMFOREST REGRESSOR ON THE FIVE NORMALIZED KPIS TO PREDICT THEIR COMPOSITE AVERAGE, USING THE RESULTING FEATURE IMPORTANCES AS THE ML-OPTIMIZED WEIGHTS.
-COMPUTED THE FINAL FDS AS THE WEIGHTED SUM AND CLASSIFIED THEM AS 'DIFFICULT', 'MEDIUM', OR 'EASY'.

```python
def calculate_flight_difficulty_score(df: pd.DataFrame) -> pd.DataFrame:
    """
    Calculates Flight Difficulty Score (FDS) using ML-optimized KPI weights.
    Uses a composite difficulty proxy as the target for RandomForest weight optimization.
    """
    if df.empty:
        print("Input DataFrame is empty. Cannot proceed with scoring.")
        return df

    # --- 1. Date/Time features ---
    df['scheduled_departure_datetime'] = pd.to_datetime(df['Schedule_Departure_Date'] + ' ' + df['Schedule_Departure_Time'])
    df['actual_departure_datetime'] = pd.to_datetime(df['Actual_Departure_Date'] + ' ' + df['Actual_Departure_Time'])

    # --- 2. Flight-Level Aggregation ---
    flight_df = df.groupby(
        ['company_id', 'flight_number', 'scheduled_departure_station_code', 'scheduled_arrival_station_code', 'Schedule_Departure_Date']
    ).agg(
        departure_delay_minutes=('actual_departure_datetime', lambda x: (x.iloc[0] - df['scheduled_departure_datetime'].iloc[df.index[x.index][0]]).total_seconds() / 60 if not x.empty else
        minimum_turn_minutes=('minimum_turn_minutes', 'first'),
        total_seats=('total_seats', 'first'),
        total_bags=('bag_type', 'count'),
        transfer_bags=('bag_type', lambda x: (x == 'Transferred').sum()),
        special_service_count=('special_service_count', 'first')
    ).reset_index()
```

```python
# Handle ratios and missing values
flight_df['transfer_bag_ratio'] = flight_df['transfer_bags'] / flight_df['total_bags'].replace(0, 1)
flight_df['departure_delay_minutes'] = flight_df['departure_delay_minutes'].fillna(0).clip(lower=0)

#  Normalize KPIs ---
scaler = MinMaxScaler()
kpi_cols = ['departure_delay_minutes', 'minimum_turn_minutes', 'total_bags', 'transfer_bag_ratio', 'special_service_count']
flight_df[['norm_' + col for col in kpi_cols]] = scaler.fit_transform(flight_df[kpi_cols])
flight_df['norm_ground_time_inverse'] = 1 - flight_df['norm_minimum_turn_minutes']
flight_df.drop(columns=['norm_minimum_turn_minutes'], inplace=True)

#  Composite difficulty proxy for ML weight learning ---
flight_df['raw_difficulty_proxy'] = (
    flight_df['norm_departure_delay_minutes'] +
    flight_df['norm_ground_time_inverse'] +
    flight_df['norm_total_bags'] +
    flight_df['norm_transfer_bag_ratio'] +
    flight_df['norm_special_service_count']
) / 5  # equal-weight sum as initial proxy
```

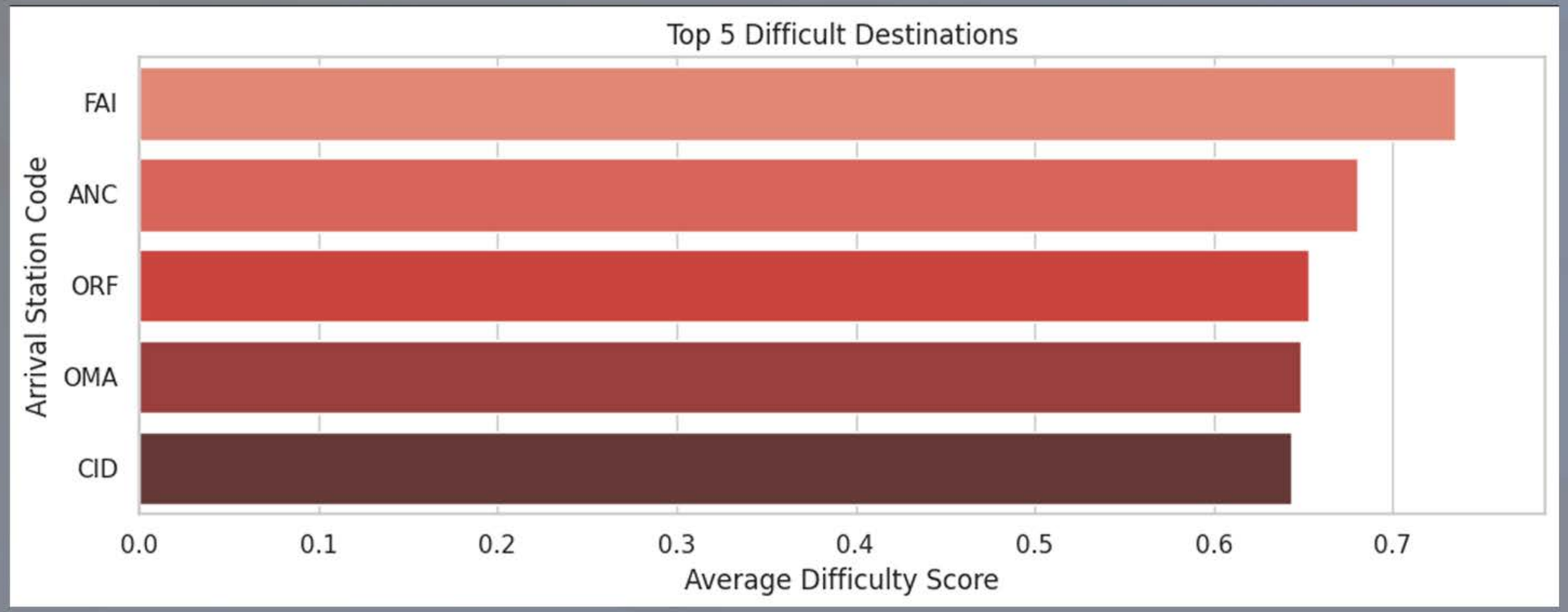

```python
#  ML-Optimized KPI Weights ---
features_for_ml = [
    'norm_departure_delay_minutes', 'norm_ground_time_inverse',
    'norm_total_bags', 'norm_transfer_bag_ratio', 'norm_special_service_count'
]
target_for_ml = 'raw_difficulty_proxy'

X = flight_df[features_for_ml].fillna(0)
y = flight_df[target_for_ml].fillna(0)

if len(flight_df) >= 10:
    rf = RandomForestRegressor(n_estimators=300, random_state=42)
    rf.fit(X, y)
    y_pred = rf.predict(X)
    print("Model R² Score:", round(r2_score(y, y_pred), 3))

    importances = rf.feature_importances_
    weights = importances / importances.sum()  # normalize

    weight_df = pd.DataFrame({
        'KPI': features_for_ml,
        'Weight': weights
    }).sort_values(by='Weight', ascending=False)

    print("\nOptimized KPI Weights (ML-based):")
    print(weight_df.to_markdown(index=False))
else:
    weights = np.ones(5) / 5
    weight_df = pd.DataFrame({'KPI': features_for_ml, 'Weight': weights})
    print("Not enough data for ML optimization; using equal weights.")

W = dict(zip(weight_df['KPI'], weight_df['Weight']))
```



```python
#  ML-Optimized KPI Weights ---
features_for_ml = [
    'norm_departure_delay_minutes', 'norm_ground_time_inverse',
    'norm_total_bags', 'norm_transfer_bag_ratio', 'norm_special_service_count'
]
target_for_ml = 'raw_difficulty_proxy'

X = flight_df[features_for_ml].fillna(0)
y = flight_df[target_for_ml].fillna(0)

if len(flight_df) >= 10:
    rf = RandomForestRegressor(n_estimators=300, random_state=42)
    rf.fit(X, y)
    y_pred = rf.predict(X)
    print("Model R² Score:", round(r2_score(y, y_pred), 3))

    importances = rf.feature_importances_
    weights = importances / importances.sum()  # normalize

    weight_df = pd.DataFrame({
        'KPI': features_for_ml,
        'Weight': weights
    }).sort_values(by='Weight', ascending=False)

    print("\nOptimized KPI Weights (ML-based):")
    print(weight_df.to_markdown(index=False))
else:
    weights = np.ones(5) / 5
    weight_df = pd.DataFrame({'KPI': features_for_ml, 'Weight': weights})
    print("Not enough data for ML optimization; using equal weights.")

W = dict(zip(weight_df['KPI'], weight_df['Weight']))
```

```python
#  Compute Difficulty Score using ML weights ---
flight_df['difficulty_score'] = (
    W['norm_departure_delay_minutes'] * flight_df['norm_departure_delay_minutes'] +
    W['norm_ground_time_inverse'] * flight_df['norm_ground_time_inverse'] +
    W['norm_total_bags'] * flight_df['norm_total_bags'] +
    W['norm_transfer_bag_ratio'] * flight_df['norm_transfer_bag_ratio'] +
    W['norm_special_service_count'] * flight_df['norm_special_service_count']
)

#  Daily Ranking & Classification ---
flight_df['flights_per_day'] = flight_df.groupby('Schedule_Departure_Date')['flight_number'].transform('count')
flight_df['difficulty_rank'] = flight_df.groupby('Schedule_Departure_Date')['difficulty_score'].rank(method='min', ascending=False).astype(int)

def classify_difficulty(row):
    total = row['flights_per_day']
    rank = row['difficulty_rank']
    if rank <= total * 0.10:
        return 'Difficult'
    elif rank <= total * 0.50:
        return 'Medium'
    else:
        return 'Easy'

flight_df['difficulty_classification'] = flight_df.apply(classify_difficulty, axis=1)

output_cols = [
    'Schedule_Departure_Date', 'company_id', 'flight_number',
    'scheduled_departure_station_code', 'scheduled_arrival_station_code',
    'difficulty_score', 'difficulty_rank', 'difficulty_classification',
    'departure_delay_minutes', 'total_seats', 'total_bags', 'transfer_bag_ratio',
    'special_service_count', 'minimum_turn_minutes'
]
```

```python
# Calculate FDS ---
df_scored = calculate_flight_difficulty_score(df_merged)

if not df_scored.empty:
    df_scored.to_csv(OUTPUT_FILE, index=False)
    print(f"\n--- Pipeline Complete ---\nScored data saved to '{OUTPUT_FILE}'")

    # Top 5 Difficult Destinations ---
    df_dest_summary = df_scored.groupby('scheduled_arrival_station_code').agg(
        total_flights=('company_id', 'count'),
        average_score=('difficulty_score', 'mean'),
        median_score=('difficulty_score', 'median'),
        max_score=('difficulty_score', 'max')
    ).reset_index()

    df_dest_summary = df_dest_summary[df_dest_summary['total_flights'] >= 10]
    top_destinations = df_dest_summary.sort_values(by=['average_score', 'max_score'], ascending=False).head(5)
    print("\nTop 5 Destination Airports by Difficulty:\n")
    print(top_destinations.round(3).to_markdown(index=False))
```



Top 5 Destination Airports by Difficulty:

scheduled_arrival_station_code	total_flights	average_score	median_score	max_score
FAI	15	0.735	0.748	0.817
ANC	28	0.68	0.676	0.756
ORF	24	0.653	0.663	0.805
OMA	61	0.649	0.66	0.78
CID	49	0.643	0.649	0.745

Overall Difficulty Classification Counts:

Classification	Count
Easy	3019
Medium	2418
Difficult	598

```
--- Top 5 Difficult Flights per Date ---
```

Date: 2025-08-01

Schedule_Departure_Date	company_id	flight_number	scheduled_arrival_station_code	difficulty_score	difficulty_rank
2025-08-01	UA	1531	DSM	0.829421	1
2025-08-01	UA	1269	CLE	0.812158	2
2025-08-01	UA	1710	SMF	0.800128	3
2025-08-01	UA	2172	FAI	0.797472	4
2025-08-01	UA	1492	CMH	0.77613	5

Date: 2025-08-02

Schedule_Departure_Date	company_id	flight_number	scheduled_arrival_station_code	difficulty_score	difficulty_rank
2025-08-02	UA	310	MCI	0.839788	1
2025-08-02	UA	1269	CLE	0.833488	2
2025-08-02	UA	1612	MEM	0.792083	3
2025-08-02	UA	1710	SMF	0.792067	4
2025-08-02	UA	1616	GRR	0.785406	5

Date: 2025-08-03

Schedule_Departure_Date	company_id	flight_number	scheduled_arrival_station_code	difficulty_score	difficulty_rank
2025-08-03	UA	310	MCI	0.813816	1
2025-08-03	UA	1492	CMH	0.775861	2
2025-08-03	UA	1778	OMA	0.762778	3
2025-08-03	UA	1495	TVC	0.754418	4
2025-08-03	UA	2343	IND	0.736135	5

AND SO ON.....

INSIGHTS ANALYSIS & ACTION PLANNING

SUMMARIZE WHICH DESTINATIONS CONSISTENTLY SHOW MORE DIFFICULTY.

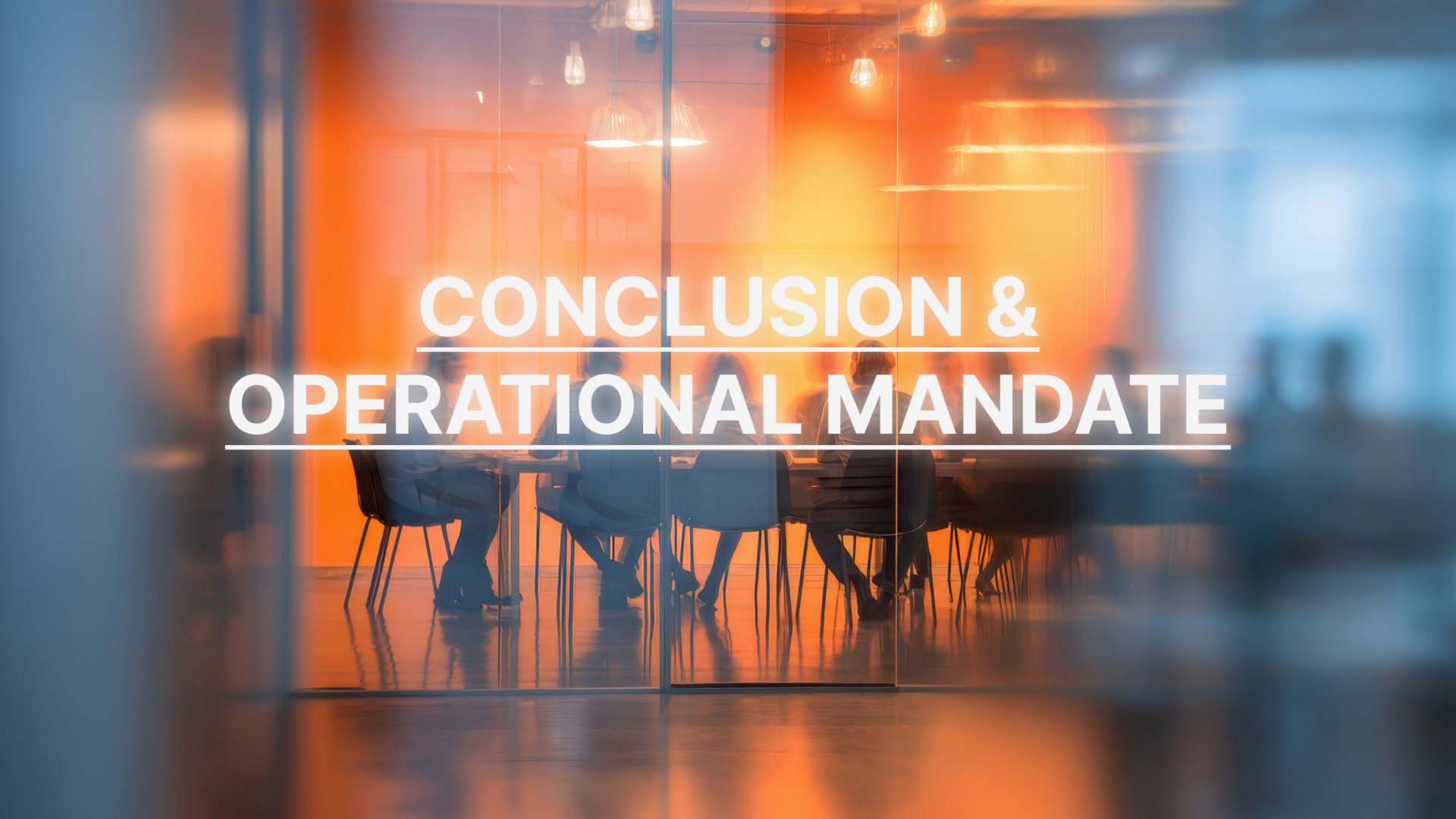
- **FAI → Consistently Extreme Difficulty:** Despite A Low Flight Volume (15), FAI Registered The Highest Maximum FDS (0.817) And The Highest Average FDS (0.735). This Indicates That Every Flight To FAI Is Likely To Involve Severe Operational Strain And Challenge.
- **ANC & ORF → High-Risk Core Destinations:** Anchorage And Norfolk Follow With Notably High Average Difficulty Scores (0.680 And 0.653). These Destinations Require Moderate To Significant Resource Allocation To Mitigate Persistent Operational Challenges.
- **OMA & CID → Challenging Medium-Volume Hubs:** Omaha And Cedar Rapids, While Handling A Higher Volume Of Total Flights, Still Maintain Elevated Average FDS Levels (0.649 And 0.643). Their Scores Confirm They Are Consistently Difficult To Manage, Suggesting A Persistent Need For Enhanced Ground Support.

WHAT ARE THE COMMON DRIVERS FOR THOSE FLIGHTS?

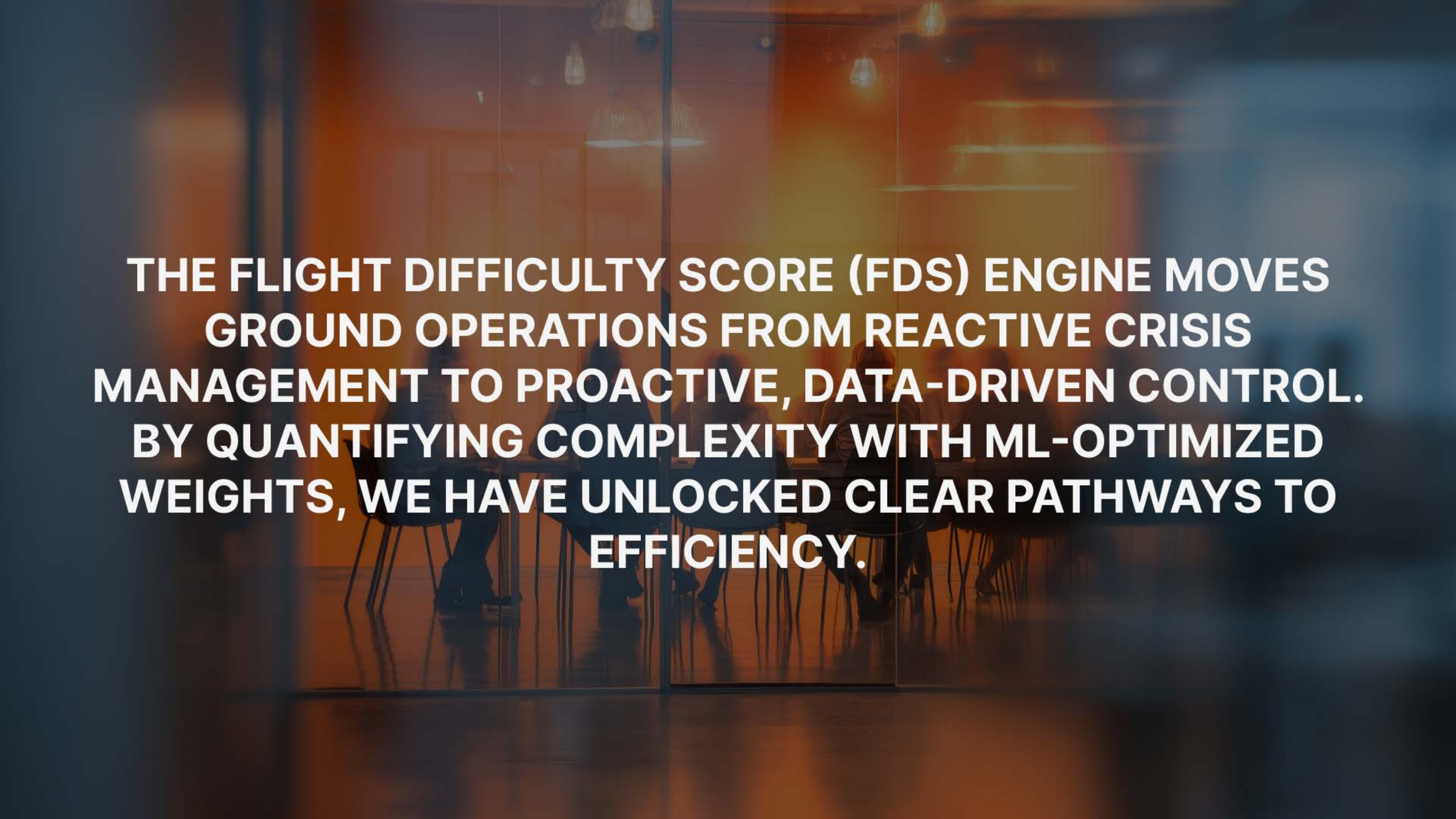
- **Operational Failure:** Departure Delay Consistently Holds A High Weight, Indicating That The Most Difficult Flights Are Simply Those That Experienced The Longest Delays.
- **Time Pressure:** Tight Turnaround Is A Critical Driver, Representing High-Stress Operations Due To Minimal Scheduled Ground Time (Low Minimum Turn Minutes).
- **Workload:** Resource Volume And Complexity—Measured By High Total Bag Count And Transfer Bag Ratio—Reflect The Substantial Physical Work And Logistical Challenge Imposed On Ground Crews.

WHAT SPECIFIC ACTIONS WOULD YOU RECOMMEND BASED ON THE FINDINGS FOR BETTER OPERATIONAL EFFICIENCY?

- **Proactive Resource Deployment:** Utilize The Daily Flight Difficulty Score (FDS) To Automatically Prioritize And Pre-Stage Resources (Staff, Equipment, Supervisors) To The Specific Flights Flagged As 'Difficult' Before Operations Begin.
- **Strategic Time Buffering:** Address The Tight Turnaround Driver By Implementing Policies To Add Time Buffers To Schedules Following Known High-Difficulty Flights, And Strategically Assign Gates To Minimize Sequential Operational Pressure.
- **Targeted Workload Management:** Reduce Complexity And Volume Strain By Deploying Dedicated Transfer Bag Teams At High-Risk Hubs (Like FAI And ANC) And Proactively Pre-Assigning Resources For High Special Service Request Flights.



CONCLUSION & OPERATIONAL MANDATE



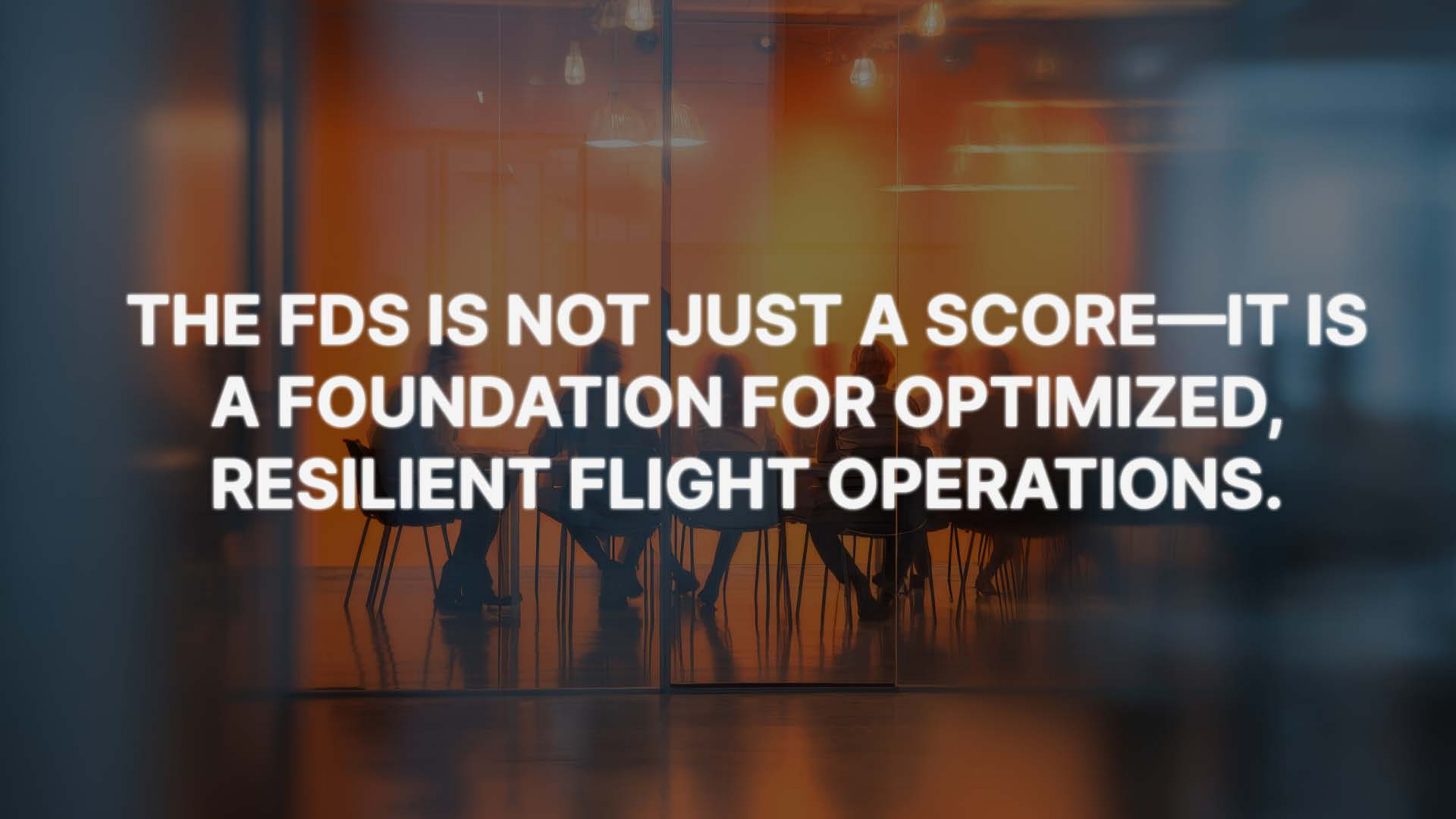
THE FLIGHT DIFFICULTY SCORE (FDS) ENGINE MOVES GROUND OPERATIONS FROM REACTIVE CRISIS MANAGEMENT TO PROACTIVE, DATA-DRIVEN CONTROL. BY QUANTIFYING COMPLEXITY WITH ML-OPTIMIZED WEIGHTS, WE HAVE UNLOCKED CLEAR PATHWAYS TO EFFICIENCY.

KEY TAKEAWAYS

- FDS IS THE NEW METRIC FOR OPERATIONAL HEALTH: THE FLIGHT DIFFICULTY SCORE SUCCESSFULLY QUANTIFIES COMPLEX GROUND STRESS POINTS BY DYNAMICALLY WEIGHTING THE TRUE DRIVERS OF OPERATIONAL FAILURE, SUCH AS TRANSFER BAG RATIO AND TIGHT TURNAROUND TIMES.
- PREDICTIVE POWER: WE HAVE IDENTIFIED AND VALIDATED THAT THE GREATEST OPERATIONAL RISK IS CONCENTRATED IN A FEW CRITICAL FACTORS (TIME PRESSURE AND LOGISTICAL WORKLOAD), ALLOWING US TO PRIORITIZE INTERVENTION BEFORE DELAYS OCCUR.
- STRATEGIC IMPERATIVE: THE ANALYSIS POINTS TO SPECIFIC, ACTIONABLE STEPS REQUIRED TO STABILIZE OPERATIONS AND REDUCE THE HIGH COSTS ASSOCIATED WITH DELAYS.

OUR RECOMMENDATION

- **PROACTIVE RESOURCE DEPLOYMENT:** UTILIZE FDS TO AUTOMATICALLY PRE-STAGE SPECIALIZED STAFF AND EQUIPMENT TO FLIGHTS FLAGGED AS 'DIFFICULT.'
- **STRATEGIC TIME BUFFERING:** SYSTEMATICALLY REVISE SCHEDULES TO ADD BUFFER TIME FOLLOWING PREDICTED HIGH-DIFFICULTY FLIGHTS, STABILIZING THE OPERATION.
- **TARGETED WORKLOAD MANAGEMENT:** DEPLOY DEDICATED TEAMS TO MANAGE TRANSFER BAGS AND HIGH SPECIAL SERVICE COUNTS AT CRITICAL AIRPORTS (FAI, ANC) TO MITIGATE LOGISTICAL STRAIN.


THE FDS IS NOT JUST A SCORE—IT IS
A FOUNDATION FOR OPTIMIZED,
RESILIENT FLIGHT OPERATIONS.

THANK YOU
TEAM
HASHIRA UNITE