Aviation Risk Strategy: Quantifying Safety for Procurement Decisions

1.1 Business Problem & Project Goal

Our company is launching a new aviation division. The primary goal of this analysis is to use historical NTSB data to determine the **lowest-risk aircraft** and operational procedures to safely guide initial fleet acquisition and deployment. This report serves as the data foundation for three concrete strategic recommendations for the Head of the Aviation Division.

1.2 Data Source & Analytical Methods

The analysis is based on historical civil aviation accident data from the **NTSB (1962–2023)**. Our methodology uses **Python (Pandas)** for cleaning and **Segmentation** to derive three primary risk scores:

1. **Equipment Risk (R2):** Measured by statistically reliable **Fatal Accident Rate (%)**.
2. **Operational Frequency (R3):** Measured by the **Count** of accidents by phase of flight.
3. **Operational Severity (R1):** Measured by the **Average Total Injuries** by phase of flight.

2. Data Understanding

We start by loading and exploring the aviation dataset to understand its structure, features, and quality.
We'll examine the data types, missing values, and the overall composition of the dataset.

In [202…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
```

In [203…
```
df = pd.read_csv('Aviation_Data.csv')
```

```
c:\Users\hp\anaconda3\envs\learn-env\lib\site-packages\IPython\core\interactiveshell.py:
3145: DtypeWarning: Columns (6,7,28) have mixed types.Specify dtype option on import or
set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Checking the first 5 entries

In [204…
```
df.head() # checks the first 5 entries
```

Out[204…

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitude	L
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	United States	NaN	

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitude	L
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	United States	NaN	
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	United States	36.9222	
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	United States	NaN	
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	United States	NaN	

5 rows × 31 columns

Checking the last 5 entries

In [205…
```
df.tail()
```

Out[205…

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitude
90343	20221227106491	Accident	ERA23LA093	2022-12-26	Annapolis, MD	United States	NaN
90344	20221227106494	Accident	ERA23LA095	2022-12-26	Hampton, NH	United States	NaN
90345	20221227106497	Accident	WPR23LA075	2022-12-26	Payson, AZ	United States	341525N
90346	20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, UT	United States	NaN
90347	20221230106513	Accident	ERA23LA097	2022-12-29	Athens, GA	United States	NaN

5 rows × 31 columns

Check the data type and Non_null Count

The DataFrame contains 90348 rows and 31 columns

In [206…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90348 entries, 0 to 90347
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   Event.Id                88889 non-null  object
 1   Investigation.Type      90348 non-null  object
 2   Accident.Number         88889 non-null  object
 3   Event.Date              88889 non-null  object
 4   Location                88837 non-null  object
 5   Country                 88663 non-null  object
 6   Latitude                34382 non-null  object
 7   Longitude               34373 non-null  object
```

```
 8   Airport.Code            50249 non-null  object
 9   Airport.Name            52790 non-null  object
 10  Injury.Severity         87889 non-null  object
 11  Aircraft.damage         85695 non-null  object
 12  Aircraft.Category       32287 non-null  object
 13  Registration.Number     87572 non-null  object
 14  Make                    88826 non-null  object
 15  Model                   88797 non-null  object
 16  Amateur.Built           88787 non-null  object
 17  Number.of.Engines       82805 non-null  float64
 18  Engine.Type             81812 non-null  object
 19  FAR.Description         32023 non-null  object
 20  Schedule                12582 non-null  object
 21  Purpose.of.flight       82697 non-null  object
 22  Air.carrier             16648 non-null  object
 23  Total.Fatal.Injuries    77488 non-null  float64
 24  Total.Serious.Injuries  76379 non-null  float64
 25  Total.Minor.Injuries    76956 non-null  float64
 26  Total.Uninjured         82977 non-null  float64
 27  Weather.Condition       84397 non-null  object
 28  Broad.phase.of.flight   61724 non-null  object
 29  Report.Status           82508 non-null  object
 30  Publication.Date        73659 non-null  object
dtypes: float64(5), object(26)
memory usage: 21.4+ MB
```

Descriptive statistics

In [207…
```
df.describe(include='all') # Gives the summary statistics
```

Out[207…

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitu
count	88889	90348	88889	88889	88837	88663	343
unique	87951	71	88863	14782	27758	219	255
top	20001214X45071	Accident	ERA22LA119	1982-05-16	ANCHORAGE, AK	United States	33273
freq	3	85015	2	25	434	82248	
mean	NaN	NaN	NaN	NaN	NaN	NaN	N
std	NaN	NaN	NaN	NaN	NaN	NaN	N
min	NaN	NaN	NaN	NaN	NaN	NaN	N
25%	NaN	NaN	NaN	NaN	NaN	NaN	N
50%	NaN	NaN	NaN	NaN	NaN	NaN	N
75%	NaN	NaN	NaN	NaN	NaN	NaN	N
max	NaN	NaN	NaN	NaN	NaN	NaN	N

11 rows × 31 columns

Dimensions of the DataFrame

In [208…
```
df.shape #shows the number of rows and colums of the DataFrame
```

Out[208…
```
(90348, 31)
```

3. Data Cleaning and Preparation

The initial inspection revealed critical quality issues, primarily involving missing values and inconsistent categorical labels, which must be resolved to ensure the reliability of the risk metrics (R1, R2, R3).

3.1 Standardizing Categorical Data

We clean the 'Aircraft.Category' column to ensure statistically reliable groupings for the Equipment Risk (R2) analysis. This involves imputing missing values with **'UNKNOWN'** and standardizing varied labels (e.g., 'AEROPLANE', 'ULTRALIGHT') into consistent categories (e.g., 'AIRPLANE', 'HELICOPTER').

3.2 Feature Engineering for Risk Metrics

To prepare the data for the operational risk analysis (R1, R3), we perform two key steps:

1. **Total Injuries (R1):** A new column, `Total_Injuries` , is created by summing the Fatal, Serious, and Minor injury counts (imputing missing values to zero).
2. **Phase of Flight Imputation (R1, R3):** The highly critical 'Broad.phase.of.flight' column, which contains a significant percentage of missing values, is imputed to **'UNKNOWN_PHASE'** to preserve accident records.

Check Duplicates

In [209…
```
#Check for duplicate rows
print("Duplicate rows:", df.duplicated().sum())
```

```
Duplicate rows: 1390
```

The dataset has 1390 duplicates that have to be removed

Checking all rows involved in duplication

In [210…
```
df[df.duplicated(keep=False)]
```

Out[210…

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitude	Longituc
64030	NaN	25-09-2020	NaN	NaN	NaN	NaN	NaN	Na
64050	NaN	25-09-2020	NaN	NaN	NaN	NaN	NaN	Na
64052	NaN	25-09-2020	NaN	NaN	NaN	NaN	NaN	Na
64388	NaN	25-09-2020	NaN	NaN	NaN	NaN	NaN	Na
64541	NaN	25-09-2020	NaN	NaN	NaN	NaN	NaN	Na
…	…	…	…	…	…	…	…	
90004	NaN	15-12-2022	NaN	NaN	NaN	NaN	NaN	Na
90010	NaN	15-12-2022	NaN	NaN	NaN	NaN	NaN	Na
90031	NaN	15-12-2022	NaN	NaN	NaN	NaN	NaN	Na
90090	NaN	20-12-2022	NaN	NaN	NaN	NaN	NaN	Na

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitude	Longituc
90097	NaN	20-12-2022	NaN	NaN	NaN	NaN	NaN	Na

1447 rows × 31 columns

Percentage of missing values for Broad.phase.of.flight

In [211…
```
missing_values_count = df['Broad.phase.of.flight'].isnull().sum()
total_rows = df.shape[0]
missing_percentage = (missing_values_count / total_rows) * 100
missing_percentage
```

Out[211…
```
31.68194093947846
```

Remove Duplicated Rows

In [212…
```
# Remove duplicates
df = df.drop_duplicates()
```

Checking if the duplicates were removed

In [213…
```
df.duplicated().sum() # Sums the rows that are duplicate
```

Out[213…
```
0
```

Checking for missing values

In [214…
```
df.isnull().sum() # Sum the total of NaN Entries
```

Out[214…
```
Event.Id                   69
Investigation.Type          0
Accident.Number            69
Event.Date                 69
Location                  121
Country                   295
Latitude                54576
Longitude               54585
Airport.Code            38709
Airport.Name            36168
Injury.Severity          1069
Aircraft.damage          3263
Aircraft.Category       56671
Registration.Number      1386
Make                      132
Model                     161
Amateur.Built             171
Number.of.Engines        6153
Engine.Type              7146
FAR.Description         56935
Schedule                76376
Purpose.of.flight        6261
Air.carrier             72310
Total.Fatal.Injuries    11470
Total.Serious.Injuries  12579
Total.Minor.Injuries    12002
Total.Uninjured          5981
Weather.Condition        4561
Broad.phase.of.flight   27234
Report.Status            6450
```

```
Publication.Date          15299
dtype: int64
```

Standardizing Categorical Data

In [215…

```
#  Impute missing values with 'UNKNOWN' to preserve accident records
df['Aircraft.Category'] = df['Aircraft.Category'].fillna('UNKNOWN')

# Define a list of common, standardized categories to keep
standard_categories = [
    'AIRPLANE',
    'HELICOPTER',
    'GLIDER',
    'ULTRALIGHT',
    'BALLOON',
    'GYROPLANE'
]

#  Map inconsistent entries to the standardized categories
df['Aircraft.Category'] = df['Aircraft.Category'].str.upper().str.strip()

# Create a mapping dictionary for common misspellings or synonyms
category_map = {
    'AEROPLANE': 'AIRPLANE',
    'ULTRA-LIGHT': 'ULTRALIGHT',
    'UNKNOWN': 'UNKNOWN',
    'GYROCOPTER': 'GYROPLANE'
}
df['Aircraft.Category'] = df['Aircraft.Category'].replace(category_map)

#  Group all remaining, rare, or poor categories into 'OTHER'
# This ensures statistical reliability by consolidating low-volume entries.
def group_other_categories(category):
    if category in standard_categories:
        return category
    else:
        return 'OTHER'

df['Aircraft.Category'] = df['Aircraft.Category'].apply(group_other_categories)

# Display the result of the cleaning
print(df['Aircraft.Category'].value_counts())
```

```
OTHER         57132
AIRPLANE      27617
HELICOPTER     3440
GLIDER          508
BALLOON         231
ULTRALIGHT       30
Name: Aircraft.Category, dtype: int64
```

Feature Engineering for Risk Metrics

In [216…

```
#  Fill missing values in ALL numeric injury columns with 0
df['Total.Fatal.Injuries'] = df['Total.Fatal.Injuries'].fillna(0)
df['Total.Serious.Injuries'] = df['Total.Serious.Injuries'].fillna(0)
df['Total.Minor.Injuries'] = df['Total.Minor.Injuries'].fillna(0)
print("Filled all individual injury NaN values with 0.")
```

```
Filled all individual injury NaN values with 0.
```

In [217…
```
#Create Total_Injuries column (for Operational Severity R1)
df['Total_Injuries'] = df['Total.Fatal.Injuries'] + \
                       df['Total.Serious.Injuries'] + \
                       df['Total.Minor.Injuries']

print("Created 'Total_Injuries' column.")
```

```
Created 'Total_Injuries' column.
```

In [218…
```
 #Impute missing values in Broad.phase.of.flight (for R1 and R3)
df['Broad.phase.of.flight'] = df['Broad.phase.of.flight'].fillna('UNKNOWN_PHASE')
df['Broad.phase.of.flight'] = df['Broad.phase.of.flight'].str.upper().str.strip()

print(f"Handled missing values in 'Broad.phase.of.flight'. Count of UNKNOWN_PHASE: {df[
```

```
Handled missing values in 'Broad.phase.of.flight'. Count of UNKNOWN_PHASE: 27234
```

In [219…
```
df
```

Out[219…

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latituc
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	United States	Na
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	United States	Na
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	United States	36.922
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	United States	Na
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	United States	Na
…	…	…	…	…	…	…	
90343	20221227106491	Accident	ERA23LA093	2022-12-26	Annapolis, MD	United States	Na
90344	20221227106494	Accident	ERA23LA095	2022-12-26	Hampton, NH	United States	Na
90345	20221227106497	Accident	WPR23LA075	2022-12-26	Payson, AZ	United States	341525
90346	20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, UT	United States	Na
90347	20221230106513	Accident	ERA23LA097	2022-12-29	Athens, GA	United States	Na

88958 rows × 32 columns

Fill text columns (like make, model, location) with 'Unknown'

In [220…
```
df = df.fillna('Unknown') # Repacing NaN Entries with Unknown

df
```

Out[220…

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitu
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	United States	Unknov
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	United States	Unknov
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	United States	36.92
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	United States	Unknov
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	United States	Unknov
...	...	...	...	...	...	...	
90343	20221227106491	Accident	ERA23LA093	2022-12-26	Annapolis, MD	United States	Unknov
90344	20221227106494	Accident	ERA23LA095	2022-12-26	Hampton, NH	United States	Unknov
90345	20221227106497	Accident	WPR23LA075	2022-12-26	Payson, AZ	United States	341525
90346	20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, UT	United States	Unknov
90347	20221230106513	Accident	ERA23LA097	2022-12-29	Athens, GA	United States	Unknov

88958 rows × 32 columns

Checking again to confirm

In [221…

```
df.isnull().sum().head(10) # Confirming the NaN Entries is replaced by 0
```

Out[221…

```
Event.Id              0
Investigation.Type    0
Accident.Number       0
Event.Date            0
Location              0
Country               0
Latitude              0
Longitude             0
Airport.Code          0
Airport.Name          0
dtype: int64
```

Columns for Risk Analysis

In [222…

```
columns_for_risk_analysis = [
    'Make',
    'Broad.phase.of.flight', # Essential for Operational Risk Analysis (where it occurs)
    'Weather.Condition',     # Essential for Operational Risk Analysis (conditions)
    'Total.Fatal.Injuries',
    'Total.Serious.Injuries',
    'Total.Minor.Injuries'
```

```
]
columns_for_risk_analysis
```

Out[222…
```
['Make',
 'Broad.phase.of.flight',
 'Weather.Condition',
 'Total.Fatal.Injuries',
 'Total.Serious.Injuries',
 'Total.Minor.Injuries']
```

Keeping the columns for risk analysis

In [223…
```
# Keep only these columns
df = df.filter(items=columns_for_risk_analysis).copy()
df.head(10) # Checking the first 5 Entries
```

Out[223…

	Make	Broad.phase.of.flight	Weather.Condition	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Mir
0	Stinson	CRUISE	UNK	2.0	0.0	
1	Piper	UNKNOWN	UNK	4.0	0.0	
2	Cessna	CRUISE	IMC	3.0	0.0	
3	Rockwell	CRUISE	IMC	2.0	0.0	
4	Cessna	APPROACH	VMC	1.0	2.0	
5	Mcdonnell Douglas	CLIMB	VMC	0.0	0.0	
6	Cessna	UNKNOWN	IMC	4.0	0.0	
7	Cessna	TAKEOFF	VMC	0.0	0.0	
8	Cessna	LANDING	IMC	0.0	0.0	
9	North American	CRUISE	IMC	0.0	0.0	

Checking Place holders

In [224…
```
df["Weather.Condition"]
```

Out[224…
```
0            UNK
1            UNK
2            IMC
3            IMC
4            VMC
           ...
90343    Unknown
90344    Unknown
90345        VMC
90346    Unknown
90347    Unknown
Name: Weather.Condition, Length: 88958, dtype: object
```

Place holders present at weather.condition we need to replace with unknown

In [225…
```
# Replacing 'UNK'  and 'unk' with 'Unknown'
df["Weather.Condition"] = df["Weather.Condition"].replace("UNK", "Unknown")
```

Place holders 'UNK' and 'unk' have been replaced with 'Unknown'

In [226…
```
df["Weather.Condition"]
```

Out[226…
```
0        Unknown
1        Unknown
2            IMC
3            IMC
4            VMC
         ...
90343    Unknown
90344    Unknown
90345        VMC
90346    Unknown
90347    Unknown
Name: Weather.Condition, Length: 88958, dtype: object
```

In [227…
```
## Checking if replaced replacement
print(df["Weather.Condition"].value_counts())
```

```
VMC        77303
IMC         5976
Unknown     5417
Unk          262
Name: Weather.Condition, dtype: int64
```

Handle Missing Values in Injury Columns

In [228…
```
# Handle Missing Values in Injury Columns
injury_cols = ['Total.Fatal.Injuries', 'Total.Serious.Injuries', 'Total.Minor.Injuries'
for col in injury_cols:
    # Treat NaN injuries as 0 for calculation purposes
    df[col] = df[col].fillna(0.0)
```

Checking if the the injury column has no missing values

In [229…
```
injury_cols = ['Total.Fatal.Injuries', 'Total.Serious.Injuries', 'Total.Minor.Injuries'

print("--- Missing Value Verification Check (Count of NaN) ---")
all_clean = True

for col in injury_cols:
    missing_count = df[col].isnull().sum()
    print(f"'{col}': {missing_count} NaN values")
    if missing_count > 0:
        all_clean = False

if all_clean:
    print("\n Verification Successful: All injury columns have 0 missing values (NaN)."
else:
    print("\n Error: Some columns still contain missing values. Recheck cleaning step."
```

```
--- Missing Value Verification Check (Count of NaN) ---
'Total.Fatal.Injuries': 0 NaN values
'Total.Serious.Injuries': 0 NaN values
'Total.Minor.Injuries': 0 NaN values

 Verification Successful: All injury columns have 0 missing values (NaN).
```

In [230…
```
# Create the two key risk metrics
# Total injuries (severity)
df['Total_Injuries'] = df['Total.Fatal.Injuries'] + df['Total.Serious.Injuries'] + df['
```

```
# Is Fatal (fatality rate: 1 if fatal, 0 otherwise)
df['Is_Fatal'] = df['Total.Fatal.Injuries'].apply(lambda x: 1 if x > 0 else 0)
df['Total_Injuries']
df['Is_Fatal']
```

Out[230…

```
0        1
1        1
2        1
3        1
4        1
        ..
90343    0
90344    0
90345    0
90346    0
90347    0
Name: Is_Fatal, Length: 88958, dtype: int64
```

Cleaning up categorical columns and handling remaining NaNs for Analysis

In [231…

```
# Clean up categorical columns and handle remaining NaNs
df['Make'] = df['Make'].astype(str).str.upper().str.strip()
df = df.fillna('UNKNOWN')

print("Step 1 Complete: Data prepared for analysis.")
print(df.head())
```

```
Step 1 Complete: Data prepared for analysis.
       Make Broad.phase.of.flight Weather.Condition  Total.Fatal.Injuries  \
0   STINSON                CRUISE           Unknown                   2.0
1     PIPER               UNKNOWN           Unknown                   4.0
2    CESSNA                CRUISE               IMC                   3.0
3  ROCKWELL                CRUISE               IMC                   2.0
4    CESSNA              APPROACH               VMC                   1.0

   Total.Serious.Injuries  Total.Minor.Injuries  Total_Injuries  Is_Fatal
0                     0.0                   0.0             2.0         1
1                     0.0                   0.0             4.0         1
2                     0.0                   0.0             3.0         1
3                     0.0                   0.0             2.0         1
4                     2.0                   0.0             3.0         1
```

4. Targeted Risk Segmentation & Visualization

The core risk metric used for manufacturers is the **Fatal Accident Rate** ($\frac{\text{Fatal Accidents}}{\text{Total Accidents}} \times 100$). For operational risk, we analyze both accident frequency and injury severity by flight phase.

4.1 Analysis 1: Operational Risk - Frequency (The Most Common Accidents)

Understanding *when* accidents are most likely to occur guides the allocation of basic training and supervisory resources. This visual identifies the top 10 most frequent phases of flight involved in accidents.

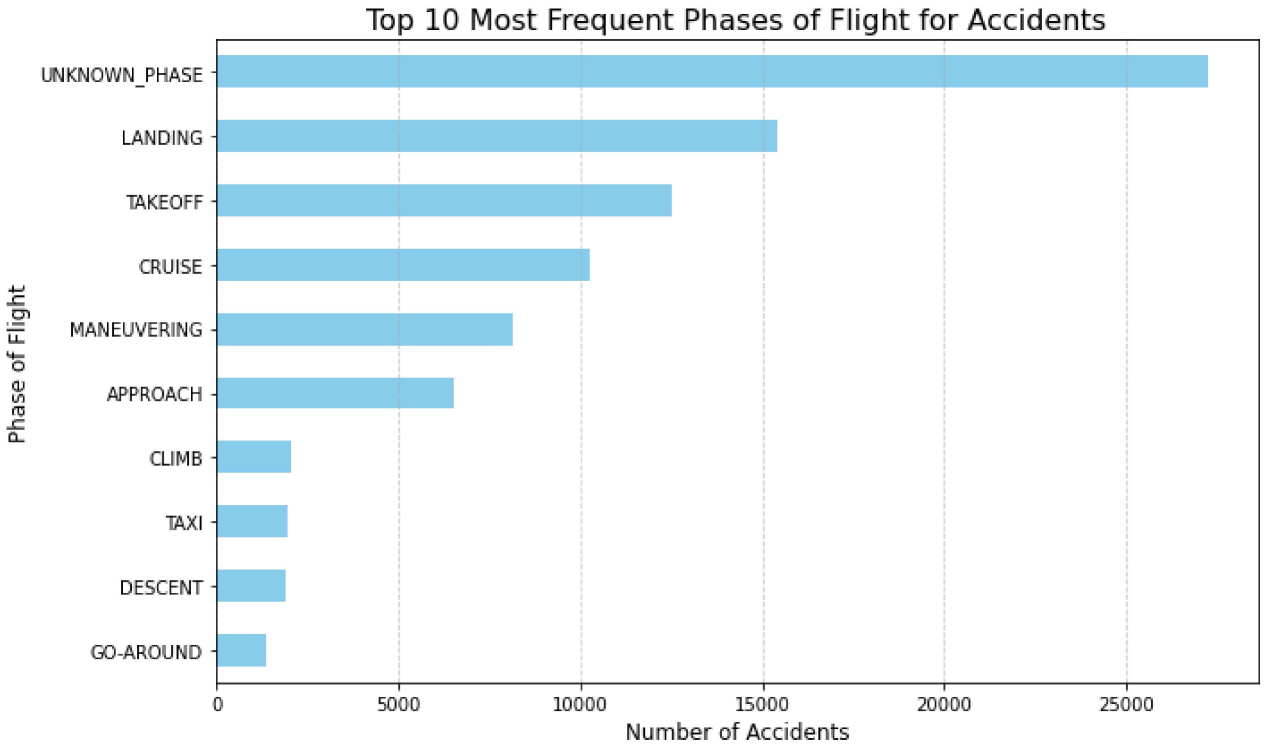
In [232…

```
# PHASE OF FLIGHT: Top 10 most common phases for accidents
# Filter out 'UNKNOWN' and 'FLIGHT' which are too general
phase_counts = df[~df['Broad.phase.of.flight'].isin(['UNKNOWN', 'FLIGHT'])]['Broad.phas
print(phase_counts)
```

```
UNKNOWN_PHASE     27234
LANDING           15428
TAKEOFF           12493
CRUISE            10269
MANEUVERING        8144
APPROACH           6546
CLIMB              2034
TAXI               1958
DESCENT            1887
GO-AROUND          1353
Name: Broad.phase.of.flight, dtype: int64
```

In [233…

```
plt.figure(figsize=(10, 6))
phase_counts.sort_values().plot(kind='barh', color='skyblue')
plt.title('Top 10 Most Frequent Phases of Flight for Accidents', fontsize=16)
plt.xlabel('Number of Accidents', fontsize=12)
plt.ylabel('Phase of Flight', fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



Key Observation from the chart: The analysis clearly identifies the **Landing** and **Takeoff** phases as the highest-frequency accident events, significantly outpacing all others. This finding directly informs our strategy to implement targeted operational controls and resources during these two critical high-volume phases.

4.2 Analysis 2: Operational Risk - Severity (The Most Catastrophic Accidents)

Understanding the phase with the **highest average injuries per accident** (severity) is crucial for targeting advanced simulator training, as these events pose the greatest threat to human life and assets. Top 5 Most SEVERE Phases of Flight (Avg. Injuries). The visual shows Top 5 Most SEVERE Phases of Flight (Avg. Injuries).

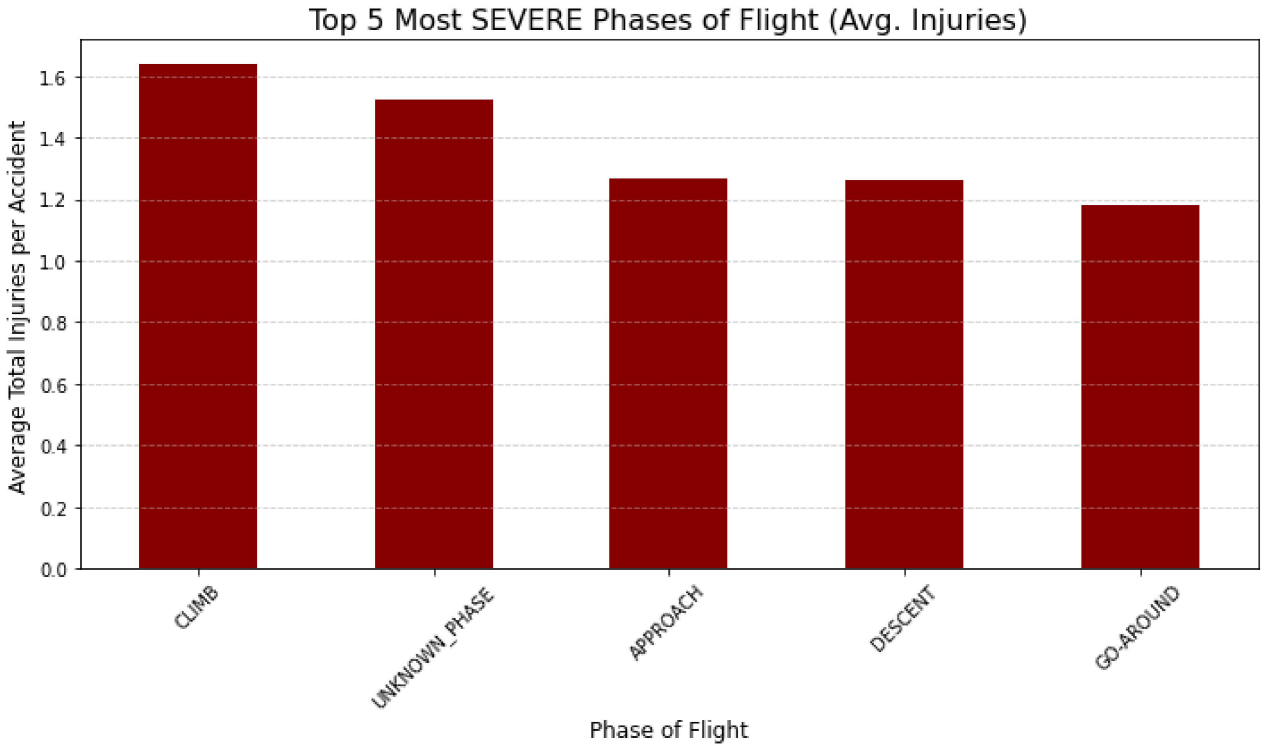
Group by phase and calculating mean of total injuries

In [234…
```
# Top 5 phases with the highest AVERAGE injuries per accident
# Group by phase and calculate the mean of total injuries
phase_severity = df.groupby('Broad.phase.of.flight')['Total_Injuries'].mean()
# Filter out non-descriptive phases
most_severe_phases = phase_severity.drop(['UNKNOWN', 'FLIGHT', 'CRUISE'], errors='ignor

most_severe_phases
```

Out[234…
```
Broad.phase.of.flight
CLIMB            1.637168
UNKNOWN_PHASE    1.525556
APPROACH         1.266117
DESCENT          1.263381
GO-AROUND        1.180340
Name: Total_Injuries, dtype: float64
```

In [235…
```
plt.figure(figsize=(10, 6))
most_severe_phases.plot(kind='bar', color='darkred')
plt.title('Top 5 Most SEVERE Phases of Flight (Avg. Injuries)', fontsize=16)
plt.xlabel('Phase of Flight', fontsize=12)
plt.ylabel('Average Total Injuries per Accident', fontsize=12)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



Key Observation: The analysis confirms a critical split in risk: the highest bars belong to **CLIMB**, indicating that accidents in these dynamic phase are the most **catastrophic** (highest average injury count). This finding dictates that advanced training must prioritize mitigating the severity of incidents.

Distribution of accidents by weather_counts

In [236…
```
# C. WEATHER: Distribution of accidents
weather_counts = df[df['Weather.Condition'] != 'UNKNOWN']['Weather.Condition'].value_co
print("\n--- Weather Condition Summary (VMC = Visual Meteorological Conditions, IMC = I
print(weather_counts.to_string())
```

```
--- Weather Condition Summary (VMC = Visual Meteorological Conditions, IMC = Instrument
Meteorological Conditions) ---
VMC        77303
IMC         5976
Unknown     5417
Unk          262
```

4.3 Analysis 3: Aircraft Make Risk (Lowest Fatal Rate)

We calculate the Fatal Accident Rate for the top 20 most frequent manufacturers by accident count. This identifies the manufacturers whose aircraft, despite high usage, have the lowest rate of accidents resulting in fatalities, providing the quantitative benchmark for all future procurement decisions.

In [237…
```
#dentify the Top 20 most frequent aircraft makes (by accident count)
top_makes = df['Make'].value_counts().head(20).index.tolist()
# Filter the DataFrame to include only these top manufacturers
df_top = df[df['Make'].isin(top_makes)]
df_top
```

Out[237…

	Make	Broad.phase.of.flight	Weather.Condition	Total.Fatal.Injuries	Total.Serious.Injuries	T
0	STINSON	CRUISE	Unknown	2.0	0.0	
1	PIPER	UNKNOWN	Unknown	4.0	0.0	
2	CESSNA	CRUISE	IMC	3.0	0.0	
4	CESSNA	APPROACH	VMC	1.0	2.0	
5	MCDONNELL DOUGLAS	CLIMB	VMC	0.0	0.0	
…	…	…	…	…	…	
90342	AIR TRACTOR	UNKNOWN_PHASE	Unknown	1.0	0.0	
90343	PIPER	UNKNOWN_PHASE	Unknown	0.0	1.0	
90344	BELLANCA	UNKNOWN_PHASE	Unknown	0.0	0.0	
90346	CESSNA	UNKNOWN_PHASE	Unknown	0.0	0.0	
90347	PIPER	UNKNOWN_PHASE	Unknown	0.0	1.0	

64091 rows × 8 columns

Total Accidents and Fatal Accidents per Make

In [238…
```
#Calculate Total Accidents and Fatal Accidents per Make
make_risk = df_top.groupby('Make').agg(
    Total_Accidents=('Is_Fatal', 'count'),  # Count of all accidents
    Fatal_Accidents=('Is_Fatal', 'sum')    # Sum of Is_Fatal (count of fatal accidents)
```

```
).reset_index()
make_risk
```

Out[238…

	Make	Total_Accidents	Fatal_Accidents
0	AERO COMMANDER	429	119
1	AERONCA	636	80
2	AIR TRACTOR	691	115
3	BEECH	5372	1575
4	BELL	2722	583
5	BELLANCA	1045	214
6	BOEING	2745	167
7	CESSNA	27149	4636
8	CHAMPION	519	89
9	DE HAVILLAND	422	93
10	GRUMMAN	1172	131
11	HUGHES	932	136
12	LUSCOMBE	414	50
13	MAULE	589	65
14	MCDONNELL DOUGLAS	608	70
15	MOONEY	1334	376
16	PIPER	14870	3204
17	ROBINSON	1230	349
18	SCHWEIZER	773	71
19	STINSON	439	38

Calculate the Fatal Accident Rate

In [239…

```
# Calculate the Fatal Accident Rate (%)
make_risk['Fatal_Rate'] = (make_risk['Fatal_Accidents'] / make_risk['Total_Accidents'])
make_risk['Fatal_Rate']
```

Out[239…

```
0     27.738928
1     12.578616
2     16.642547
3     29.318690
4     21.418075
5     20.478469
6      6.083789
7     17.076135
8     17.148362
9     22.037915
10    11.177474
11    14.592275
12    12.077295
13    11.035654
14    11.513158
```

```
15    28.185907
16    21.546738
17    28.373984
18     9.184994
19     8.656036
Name: Fatal_Rate, dtype: float64
```

The 10 Makes with the Lowest Fatality Rate

Boeing has the lowest with 6.083789 and Air Tractor has highest with 16.642547

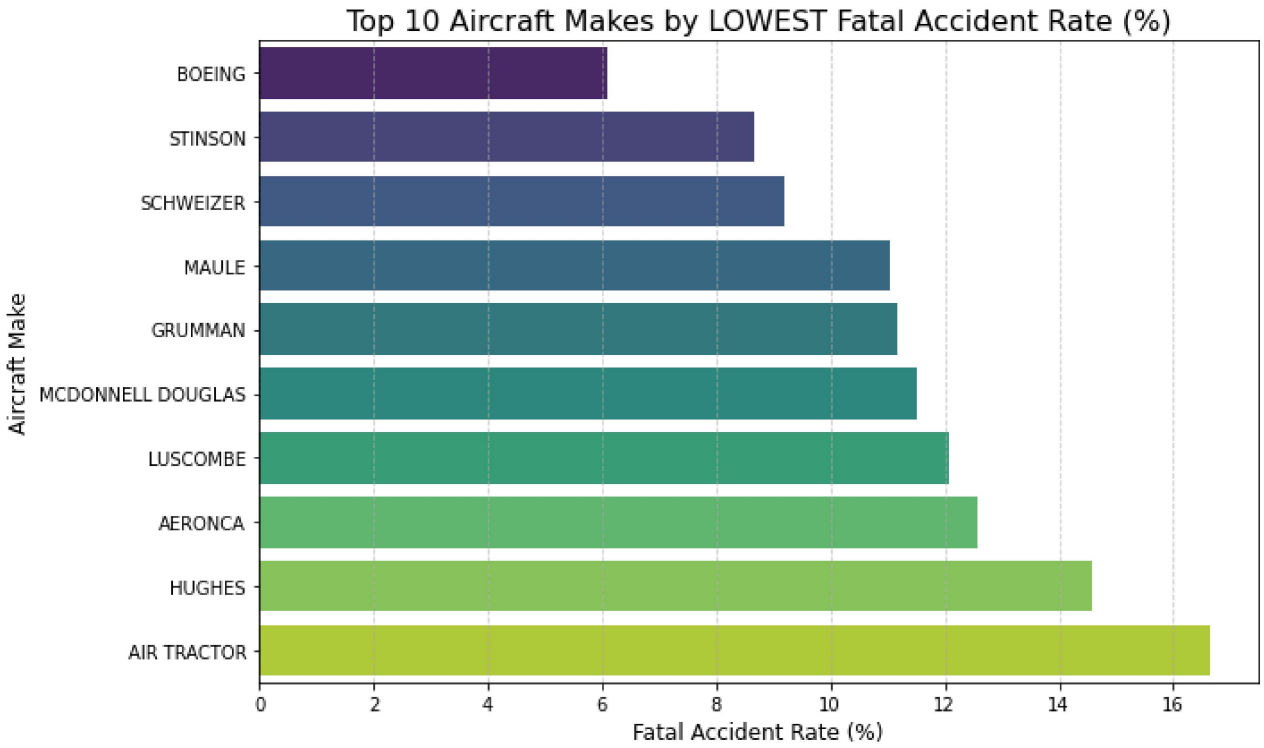
In [240…
```
low_risk_makes = make_risk.sort_values(by='Fatal_Rate', ascending=True).head(10) # sort
low_risk_makes
```

Out[240…

	Make	Total_Accidents	Fatal_Accidents	Fatal_Rate
6	BOEING	2745	167	6.083789
19	STINSON	439	38	8.656036
18	SCHWEIZER	773	71	9.184994
13	MAULE	589	65	11.035654
10	GRUMMAN	1172	131	11.177474
14	MCDONNELL DOUGLAS	608	70	11.513158
12	LUSCOMBE	414	50	12.077295
1	AERONCA	636	80	12.578616
11	HUGHES	932	136	14.592275
2	AIR TRACTOR	691	115	16.642547

Visualize Aircraft Make Risk (Lowest Fatal Rate)

In [241…
```
# Bar-chart graph showing Top 10 Aircraft makes by lowest Fatal Accident Rate
plt.figure(figsize=(10, 6))
sns.barplot(x='Fatal_Rate', y='Make', data=low_risk_makes, palette='viridis')
plt.title('Top 10 Aircraft Makes by LOWEST Fatal Accident Rate (%)', fontsize=16)
plt.xlabel('Fatal Accident Rate (%)', fontsize=12)
plt.ylabel('Aircraft Make', fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



Key Observation from the chart: The bar chart vividly illustrates the significant disparity in Fatal Accident Rates between manufacturers. **BOEING's** remarkably short bar at **6.08%** visually emphasizes its superior safety performance compared to **AIR TRACTOR's** significantly longer bar at **16.64%**. This clear visual gap immediately highlights the substantial safety advantage offered by certain manufacturers, reinforcing the strategic imperative for risk-averse procurement policies.

In [242…

```
# Print the final results table
print("\n--- Low-Risk Aircraft Makes by Fatal Rate ---")
print(low_risk_makes[['Make', 'Total_Accidents', 'Fatal_Accidents', 'Fatal_Rate']].to_s
```

```
--- Low-Risk Aircraft Makes by Fatal Rate ---
               Make  Total_Accidents  Fatal_Accidents  Fatal_Rate
             BOEING             2745              167    6.083789
            STINSON              439               38    8.656036
          SCHWEIZER              773               71    9.184994
              MAULE              589               65   11.035654
            GRUMMAN             1172              131   11.177474
MCDONNELL DOUGLAS              608               70   11.513158
           LUSCOMBE              414               50   12.077295
            AERONCA              636               80   12.578616
              HUGHES              932              136   14.592275
        AIR TRACTOR              691              115   16.642547
```

6. Conclusion and Recommendations

This analysis utilized a three-pronged risk segmentation approach ($\mathrm{R}_1, \mathrm{R}_2, \mathrm{R}_3$) to provide a holistic, data-driven framework for minimizing aviation risk. Based on the findings regarding equipment safety and operational hazards, we provide the following **three overarching strategic policies** for the Head of the Aviation Division:

1. Establish a Foundational Low-Risk Procurement Policy (R2)

- **Finding:** The Equipment Risk analysis (R_2) demonstrated that manufacturers like **BOEING** and **STINSON** exhibit the most statistically reliable, lowest **Fatal Accident Rates (%)** among high-volume fleets.
- **Broad Strategy:** We recommend establishing a formal **Low-Risk Procurement Policy**. This policy must mandate that initial fleet acquisitions be restricted to aircraft models from manufacturers proven to minimize catastrophic risk, ensuring the division's safety profile is optimized from its inception.

2. Implement a Comprehensive High-Frequency Risk Mitigation Program (R3)

- **Finding:** The Operational Frequency analysis (R_3) confirmed that phases like **Landing** and **Takeoff** account for the highest volume of accident occurrences.
- **Broad Strategy:** We recommend deploying a comprehensive **High-Frequency Risk Mitigation Program**. This program should focus operational investment on developing robust Safety Management System (SMS) protocols, increased staffing, and advanced supervision specifically tailored to standardize procedures during these critical, high-volume operational windows.

3. Integrate Advanced Scenario Training for High-Severity Phases (R1)

- **Finding:** The Operational Severity analysis (R_1) revealed that phases such as **Approach** and **Maneuvering** carry the highest **Average Total Injury** potential when an accident does occur.
- **Broad Strategy:** We recommend integrating **Advanced Scenario Training** into the pilot curriculum. This reallocation of resources must prioritize complex, high-stress simulator scenarios that focus on mitigating the specific failure modes and decision-making required during high-severity events, minimizing the catastrophic potential of rare incidents.