FATUMA A. TARI, DSF-PT12, PHASE 1 PROJECT

22nd JULY 2025,

FIDELIS WANALWENGE.

STEP 1: LOAD AND INSPECT THE DATA

We begin by loading the aviation dataset using Pandas and inspecting its structure. This helps us understand the number of records, columns, data types, and missing values. I will also load the standard libraries needed for data analysis.

We also look for potential issues like inconsistent column formats, mixed data types and irrelevant features.

```
#Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns


df = pd.read_csv('/content/Aviation_Data.csv')
df
```

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Loca
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MC CREE
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEP
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltvill
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Cantor
...	...	...	...	...	
23694	20001213X29181	Accident	MIA89LA213	1989-08-05	SARAS
23695	20001213X29151	Accident	LAX89LA264	1989-08-05	PAF DAM
23696	20001213X29102	Accident	DEN89LA191	1989-08-05	BOUL
23697	20001213X29097	Accident	DEN89LA186	1989-08-05	LOVELL
23698	20001213X29074	Accident	CHI89LA161	1989-08-05	

23699 rows × 31 columns

```
df = pd.read_csv('/content/Aviation_Data.csv', low_memory=False)
df
```

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Loca
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MC CREE
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEP
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltvill
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Cantor
...	...	...	...	...	
28424	20001212X17553	Accident	FTW91LA116	1991-07-04	LAKEV
28425	20001212X17493	Accident	CHI91LA209	1991-07-04	HILL CIT
28426	20001212X17476	Accident	CHI91DEX04	1991-07-04	LEBANO
28427	20001212X17459	Accident	BFO91LA062	1991-07-04	SC BOSTO
28428	20001212X17413	Accident	ANC91LA096	1991-07-04	SALMO

28429 rows × 31 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28429 entries, 0 to 28428
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Event.Id                28429 non-null  object 
 1   Investigation.Type      28429 non-null  object 
 2   Accident.Number         28429 non-null  object 
 3   Event.Date              28429 non-null  object 
 4   Location                28419 non-null  object 
 5   Country                 28294 non-null  object 
 6   Latitude                8 non-null      float64
 7   Longitude               8 non-null      float64
 8   Airport.Code            15056 non-null  object 
 9   Airport.Name            16756 non-null  object 
 10  Injury.Severity         28429 non-null  object 
 11  Aircraft.damage         27811 non-null  object 
 12  Aircraft.Category       3655 non-null   object 
 13  Registration.Number     28420 non-null  object 
 14  Make                    28422 non-null  object 
```

```
 15  Model                   28412 non-null  object
 16  Amateur.Built           28428 non-null  object
 17  Number.of.Engines       28103 non-null  float64
 18  Engine.Type             28425 non-null  object
 19  FAR.Description         3655 non-null   object
 20  Schedule                4652 non-null   object
 21  Purpose.of.flight       28388 non-null  object
 22  Air.carrier             1493 non-null   object
 23  Total.Fatal.Injuries    28293 non-null  float64
 24  Total.Serious.Injuries  28253 non-null  float64
 25  Total.Minor.Injuries    28249 non-null  float64
 26  Total.Uninjured         28313 non-null  float64
 27  Weather.Condition       28426 non-null  object
 28  Broad.phase.of.flight   28213 non-null  object
 29  Report.Status           28428 non-null  object
 30  Publication.Date        15983 non-null  object
dtypes: float64(7), object(24)
memory usage: 6.7+ MB
```

```
df.columns = df.columns.str.strip().str.lower().str.replace('.', '_').str.replace
df.columns
```

```
Index(['event_id', 'investigation_type', 'accident_number', 'event_date',
       'location', 'country', 'latitude', 'longitude', 'airport_code',
       'airport_name', 'injury_severity', 'aircraft_damage',
       'aircraft_category', 'registration_number', 'make', 'model',
       'amateur_built', 'number_of_engines', 'engine_type',
'far_description',
       'schedule', 'purpose_of_flight', 'air_carrier',
'total_fatal_injuries',
       'total_serious_injuries', 'total_minor_injuries', 'total_uninjured',
       'weather_condition', 'broad_phase_of_flight', 'report_status',
       'publication_date'],
      dtype='object')
```

STEP 2: DROP IRRELEVANT COLUMNS

Several columns in the dataset do not contribute meaningfully to our analysis or have excessive missing values. These include location coordinates, airport codes, carrier info, and registration details. To simplify the dataset and reduce noise.

```
df = df.drop(columns=[
    'latitude', 'longitude', 'airport_code', 'airport_name', 'far_description',
    'schedule', 'air_carrier', 'registration_number', 'publication_date'
], errors='ignore')
df
```

	event_id	investigation_type	accident_number	event_date	loca
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MC CREE
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEP
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltvill
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Cantor
...	...	...	...	...	
28424	20001212X17553	Accident	FTW91LA116	1991-07-04	LAKEV
28425	20001212X17493	Accident	CHI91LA209	1991-07-04	HILL CIT
28426	20001212X17476	Accident	CHI91DEX04	1991-07-04	LEBANO
28427	20001212X17459	Accident	BFO91LA062	1991-07-04	SC BOSTO
28428	20001212X17413	Accident	ANC91LA096	1991-07-04	SALMO

28429 rows × 22 columns

```
df.isna().any()
```

	0
event_id	False
investigation_type	False
accident_number	False
event_date	False
location	True
country	True
injury_severity	False
aircraft_damage	True
aircraft_category	True
make	True
model	True
amateur_built	True
number_of_engines	True
engine_type	True
purpose_of_flight	True
total_fatal_injuries	True
total_serious_injuries	True
total_minor_injuries	True
total_uninjured	True
weather_condition	True
broad_phase_of_flight	True
report_status	True

dtype: bool

```
df.isna().sum()
```

	0
event_id	0
investigation_type	0
accident_number	0
event_date	0
location	10
country	135
injury_severity	0
aircraft_damage	618
aircraft_category	24774
make	7
model	17
amateur_built	1
number_of_engines	326
engine_type	4
purpose_of_flight	41
total_fatal_injuries	136
total_serious_injuries	176
total_minor_injuries	180
total_uninjured	116
weather_condition	3
broad_phase_of_flight	216
report_status	1

dtype: int64

STEP 3: HANDLING MISSING DATA

Many columns in the dataset contain missing values. I'll take a strategic approach and handle them based on the column type and relevance.

```
# Make a copy of the original DataFrame to work with
df_clean = df.copy()
df_clean
```

	event_id	investigation_type	accident_number	event_date	loca
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MC CREE
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEP
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltvill
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Cantor
...	...	...	...	...	
28424	20001212X17553	Accident	FTW91LA116	1991-07-04	LAKEV
28425	20001212X17493	Accident	CHI91LA209	1991-07-04	HILL CIT
28426	20001212X17476	Accident	CHI91DEX04	1991-07-04	LEBANO
28427	20001212X17459	Accident	BFO91LA062	1991-07-04	SC BOSTOI
28428	20001212X17413	Accident	ANC91LA096	1991-07-04	SALMOI

28429 rows × 22 columns

STEP 3A: Fill categorical columns with "Unknown"

Columns like: aircraft_damage, injury_severity, weather_condition, purpose_of_flight, broad_phase_of_flight, report_status, aircraft_category, engine_type, amateur_built are labels and you can’t compute a mean or median here. Also if you impute with mode, you’ll inject bias by repeating the most common value. I will fill catergorical columns with "Unknown" because it will preserve missingness as a signal, avoid bais of mode input. This will still allow me to group data and visiualize the data but will get the bar labelled as unknown which still makes it clear ths was originally missing

```
# These are descriptive labels, I don't want to guess or inject bias

categorical_cols = [
    'aircraft_damage', 'injury_severity', 'weather_condition',
    'purpose_of_flight', 'broad_phase_of_flight', 'report_status',
    'aircraft_category', 'engine_type', 'amateur_built'
]
```

```
df_clean[categorical_cols] = df_clean[categorical_cols].fillna('Unknown')
```

STEP 3B: Fill numerical Injury Data

Columns like total_fatal_injuries, total_serious_injuries, total_minor_injuries, and total_uninjured —should will remain numeric for summing and analysis. I am assuming that missing values indicate zero injuries rather than unknowns since actual injuries would likely have been recorded.

```
# If injury counts are missing, we assume zero

injury_cols = [
    'total_fatal_injuries', 'total_serious_injuries',
    'total_minor_injuries', 'total_uninjured'
]

df_clean[injury_cols] = df_clean[injury_cols].fillna(0)
```

STEP 3C: Fill Make, Model, Country data

These are critical for grouping and analysis. But they don't have a clear default so i will fill with unknown.

If i'll be doing risk per manufacturer, i will drop the rows where these are missing.

```
# Make, Model, and Country filled with unknown

df_clean[['make', 'model', 'country']] = df_clean[['make', 'model', 'country']].f
```

STEP 3D: Drop rows where IDs or key dates are missing

These rows are not useful without identifiers or event date

```
df_clean = df_clean.dropna(subset=['event_id', 'accident_number', 'event_date'])
df_clean
```

	event_id	investigation_type	accident_number	event_date	loca
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MC CREE
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEP
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltvill
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Cantor
...	...	...	...	...	
28424	20001212X17553	Accident	FTW91LA116	1991-07-04	LAKEV
28425	20001212X17493	Accident	CHI91LA209	1991-07-04	HILL CIT
28426	20001212X17476	Accident	CHI91DEX04	1991-07-04	LEBANO
28427	20001212X17459	Accident	BFO91LA062	1991-07-04	SC BOSTO
28428	20001212X17413	Accident	ANC91LA096	1991-07-04	SALMO

28429 rows × 22 columns

```
df_clean.isna().sum()
```

	0
event_id	0
investigation_type	0
accident_number	0
event_date	0
location	10
country	0
injury_severity	0
aircraft_damage	0
aircraft_category	0
make	0
model	0
amateur_built	0
number_of_engines	326
engine_type	0
purpose_of_flight	0
total_fatal_injuries	0
total_serious_injuries	0
total_minor_injuries	0
total_uninjured	0
weather_condition	0
broad_phase_of_flight	0
report_status	0

dtype: int64

```
df_clean.head()
```

	event_id	investigation_type	accident_number	event_date	location
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH

5 rows × 22 columns

STEP 3E: Final Missing Value Handling

There is still some data missing, which i have to handle before exploratory data analysis.

```
# Fill missing locations with 'Unknown'

df_clean['location'] = df_clean['location'].fillna('Unknown')


# Fill missing number_of_engines with the median value

median_engines = df_clean['number_of_engines'].median()
df_clean['number_of_engines'] = df_clean['number_of_engines'].fillna(median_engin


df_clean.isna().sum()
```

	0
event_id	0
investigation_type	0
accident_number	0
event_date	0
location	0
country	0
injury_severity	0
aircraft_damage	0
aircraft_category	0
make	0
model	0
amateur_built	0
number_of_engines	0
engine_type	0
purpose_of_flight	0
total_fatal_injuries	0
total_serious_injuries	0
total_minor_injuries	0
total_uninjured	0
weather_condition	0
broad_phase_of_flight	0
report_status	0

dtype: int64

Convert date column to datetime

Convert the date column for time-based analysis later.

```
df_clean['event_date'] = pd.to_datetime(df_clean['event_date'], errors='coerce')
df_clean
```

	event_id	investigation_type	accident_number	event_date	loca
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MC CREE
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEP
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltvill
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Cantor
...	...	...	...	...	
28424	20001212X17553	Accident	FTW91LA116	1991-07-04	LAKEV
28425	20001212X17493	Accident	CHI91LA209	1991-07-04	HILL CIT
28426	20001212X17476	Accident	CHI91DEX04	1991-07-04	LEBANO
28427	20001212X17459	Accident	BFO91LA062	1991-07-04	SC BOSTO
28428	20001212X17413	Accident	ANC91LA096	1991-07-04	SALMO

28429 rows × 22 columns

```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28429 entries, 0 to 28428
Data columns (total 22 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   event_id                28429 non-null  object
 1   investigation_type      28429 non-null  object
 2   accident_number         28429 non-null  object
 3   event_date              28429 non-null  datetime64[ns]
 4   location                28429 non-null  object
 5   country                 28429 non-null  object
 6   injury_severity         28429 non-null  object
 7   aircraft_damage         28429 non-null  object
 8   aircraft_category       28429 non-null  object
 9   make                    28429 non-null  object
 10  model                   28429 non-null  object
 11  amateur_built           28429 non-null  object
 12  number_of_engines       28429 non-null  float64
 13  engine_type             28429 non-null  object
 14  purpose_of_flight       28429 non-null  object
 15  total_fatal_injuries    28429 non-null  float64
```

```
 16  total_serious_injuries  28429 non-null  float64
 17  total_minor_injuries    28429 non-null  float64
 18  total_uninjured         28429 non-null  float64
 19  weather_condition       28429 non-null  object
 20  broad_phase_of_flight   28429 non-null  object
 21  report_status           28429 non-null  object
dtypes: datetime64[ns](1), float64(5), object(16)
memory usage: 4.8+ MB
```

Save my cleaned data into a file named aircraft_cleaned_data.csv. to use for tableau interactive dashboard

```
df_clean.to_csv('aircraft_cleaned_data.csv', index=False)
```

Double-click (or enter) to edit

STEP 4: EXPLORATORY DATA ANALYSIS

Before i dive into the three key business questions, i will perform an exploratory data analysis (EDA) to better understand the distributions, patterns, and trends within the cleaned dataset. This will help me uncover potential risk factors, spot anomalies, and refine my approach to answering business-relevant questions.

i will focus on high-impact variables such as injury severity, aircraft damage, aircraft type, flight purpose, and time trends.

1. Injury Severity Distribution

```
df_clean['injury_severity'].value_counts()
```

injury_severity	count
Non-Fatal	22084
Fatal(1)	2638
Fatal(2)	1585
Incident	1008
Fatal(3)	478
Fatal(4)	381
Fatal(5)	78
Fatal(6)	69
Fatal(7)	23
Fatal(8)	20
Unavailable	12
Fatal(10)	8
Fatal(9)	5
Fatal(17)	3
Fatal(25)	3
Fatal(14)	3
Fatal(11)	2
Fatal(23)	2
Fatal(82)	2
Fatal(34)	2
Fatal(12)	2
Fatal(78)	1
Fatal(70)	1
Fatal(153)	1
Fatal(29)	1
Fatal(13)	1
Fatal(256)	1
Fatal(31)	1
Fatal(135)	1
Fatal(156)	1
Fatal(18)	1

Fatal(43)	1
Fatal(15)	1
Fatal(28)	1
Fatal(270)	1
Fatal(144)	1
Fatal(111)	1
Fatal(174)	1
Fatal(131)	1
Fatal(20)	1
Fatal(73)	1
Fatal(27)	1

dtype: int64

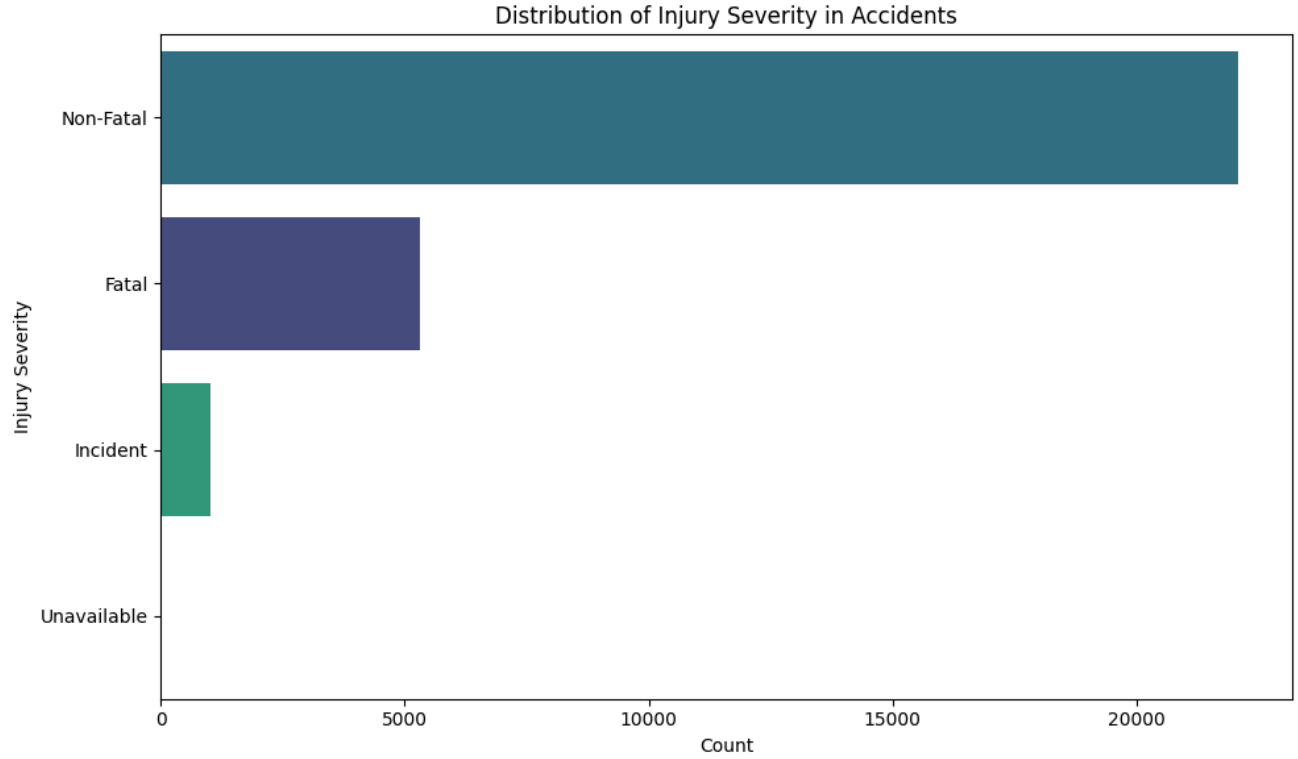
```
# Step 1: Remove rows with missing injury_severity and make a copy
df_clean = df[df['injury_severity'].isna() == False].copy()

# Step 2: Strip spaces and convert to title case
df_clean['injury_severity'] = df_clean['injury_severity'].str.strip().str.title()

# Step 3: Replace all "Fatal(x)" values with "Fatal"
df_clean['injury_severity'] = df_clean['injury_severity'].apply(
    lambda x: 'Fatal' if x.startswith('Fatal') else x
)

# Step 4: Standardize other values
df_clean['injury_severity'] = df_clean['injury_severity'].replace({
    'Non Fatal': 'Non-Fatal',
    'Incident': 'Incident',
    'Uninjured': 'Uninjured',
    'Minor': 'Minor'
})


df_clean['injury_severity'].value_counts()
```

injury_severity	count
Non-Fatal	22084
Fatal	5325
Incident	1008
Unavailable	12

dtype: int64

```
# Set figure size
plt.figure(figsize=(10, 6))

# Create a sorted countplot for injury_severity
sns.countplot(
    data=df_clean,
    y='injury_severity',
    order=df_clean['injury_severity'].value_counts().index,
    hue='injury_severity',
    palette='viridis',
    dodge=False)

# Remove legend (since y-labels already indicate the categories)
plt.legend().remove()

# Add title and labels
plt.title('Distribution of Injury Severity in Accidents')
plt.xlabel('Count')
plt.ylabel('Injury Severity')

# Improve layout
plt.tight_layout()

# Show plot
plt.show()
```

```
/tmp/ipython-input-25-2334932219.py:14: UserWarning: No artists with labels f
  plt.legend().remove()
```



Most aviation events in this dataset are non-fatal, but a significant number involve fatal outcomes. Categories like Minor, Serious, and Incident also suggest a wide spectrum of accident consequences, each with different operational implications.

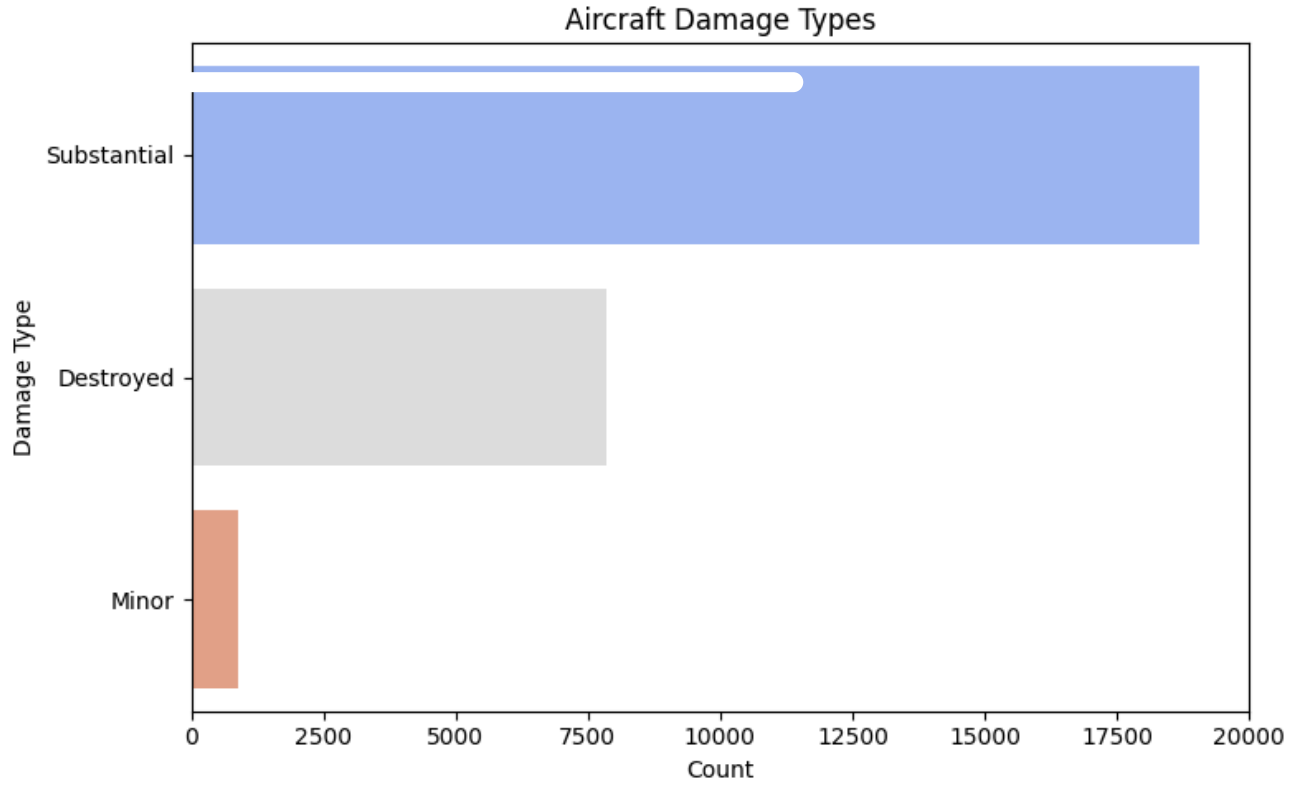
Insights: This distribution will support risk stratification, helping stakeholders understand which types of accidents occur most frequently and how to prioritize interventions accordingly.

2. Aircraft Damage Types

```
# Distribution of aircraft damage
plt.figure(figsize=(8,5))
sns.countplot(data=df_clean, y='aircraft_damage', order=df_clean['aircraft_damage
plt.title('Aircraft Damage Types')
plt.xlabel('Count')
plt.ylabel('Damage Type')
```

```
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-26-142125306.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed i

  sns.countplot(data=df_clean, y='aircraft_damage', order=df_clean['aircraft_
```



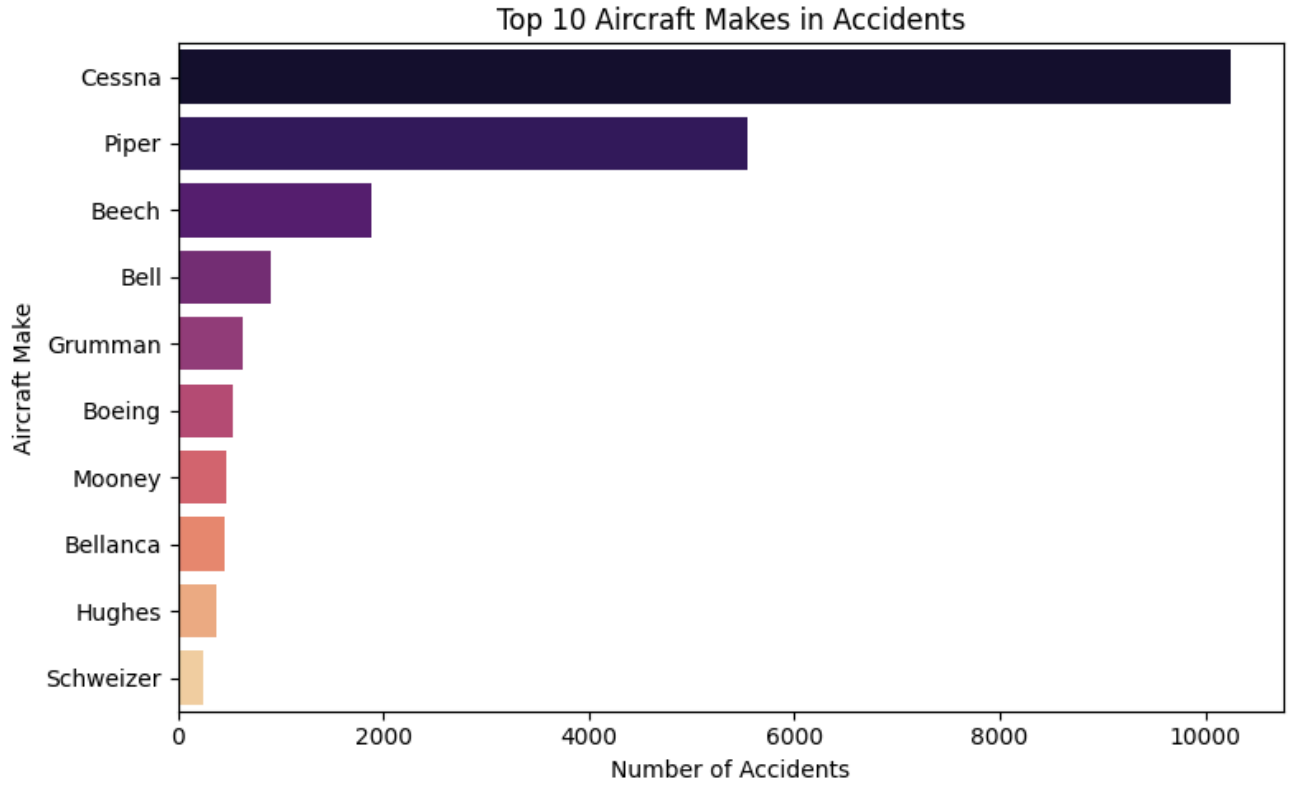
Insight: The majority of aircraft accidents result in substantial or destroyed damage, which implies high operational and repair costs. This has implications for insurance, safety policy, and operational risk management. Knowing how often each type of damage occurs can support better resource planning and help prioritize preventive maintenance or pilot training programs.

3. Top 10 Aircraft Makes

```
# Top 10 most frequent aircraft makes involved in accidents
top_makes = df_clean['make'].value_counts().head(10)

plt.figure(figsize=(8,5))
sns.barplot(x=top_makes.values, y=top_makes.index, palette='magma')
plt.title('Top 10 Aircraft Makes in Accidents')
plt.xlabel('Number of Accidents')
plt.ylabel('Aircraft Make')
```

```
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-27-718277602.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed i

  sns.barplot(x=top_makes.values, y=top_makes.index, palette='magma')
```

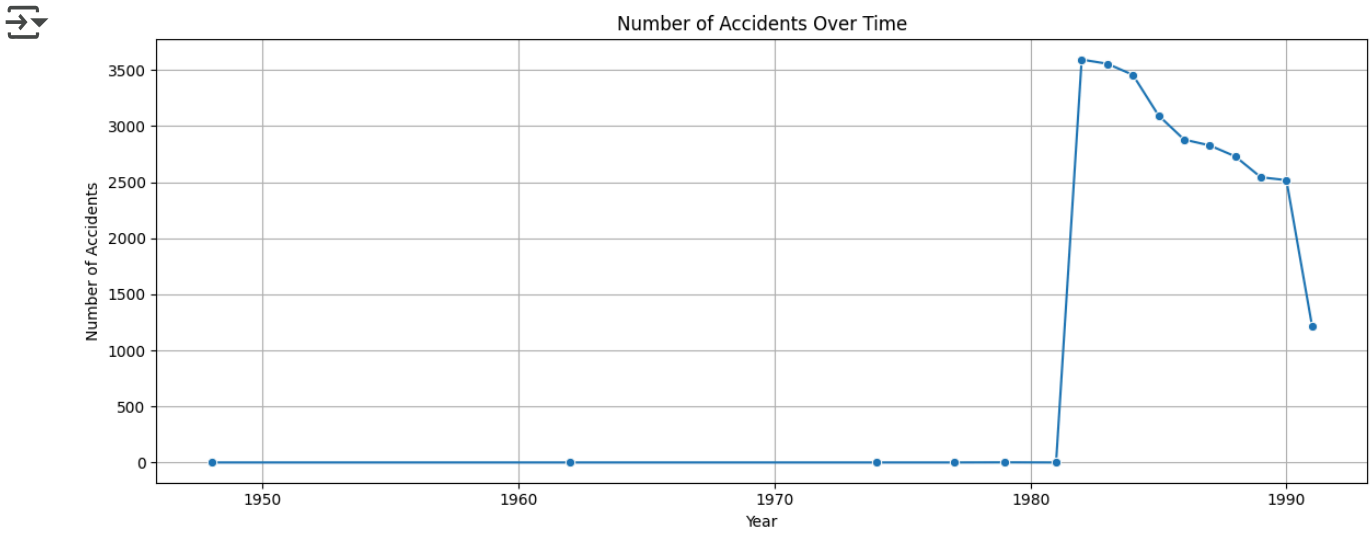


insight: Understanding which aircraft makes appear most often in accident reports can inform maintenance protocols, training standards, or manufacturer-specific safety reviews. It may also guide decisions in procurement or fleet modernization for aviation organizations evaluating safety performance.

5. Accidents Over Time

```
# Convert event_date to datetime format and extract year
df_clean['event_date'] = pd.to_datetime(df_clean['event_date'], errors='coerce')
df_clean['year'] = df_clean['event_date'].dt.year

# Plot accidents per year
accidents_per_year = df_clean['year'].value_counts().sort_index()

plt.figure(figsize=(12,5))
sns.lineplot(x=accidents_per_year.index, y=accidents_per_year.values, marker='o')
```

```
plt.title('Number of Accidents Over Time')
plt.xlabel('Year')
plt.ylabel('Number of Accidents')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Tracking accident trends over time enables aviation authorities and organizations to evaluate the impact of safety interventions, allocate resources efficiently, and forecast future safety needs.

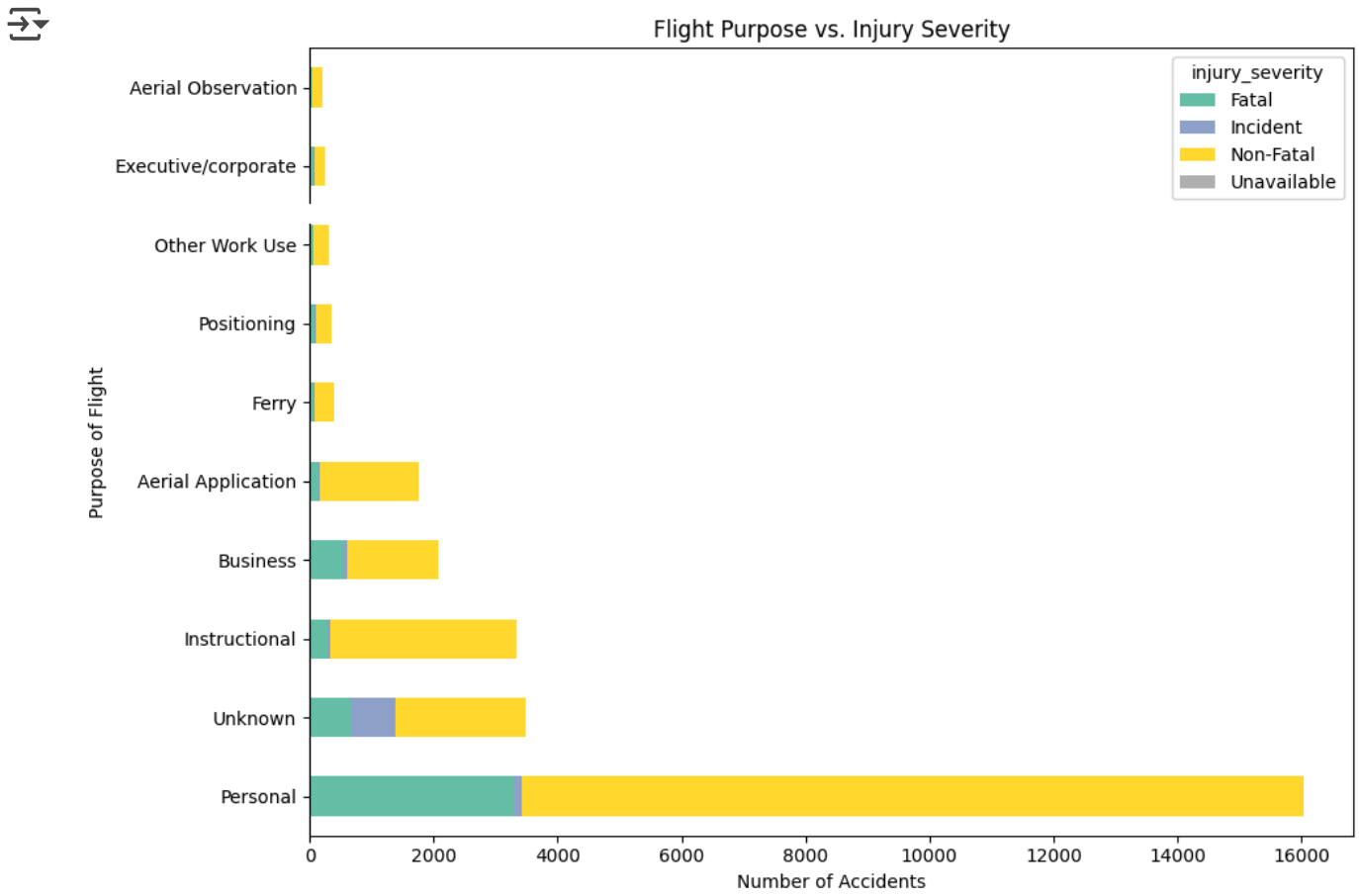
A sharp spike after 1980 could reflect reporting issues before that year.

Insight: Tracking accident trends over time enables aviation authorities and organizations to evaluate the impact of safety interventions, allocate resources efficiently, and forecast future safety needs.

6. Flight Purpose vs. Injury Severity

```
# Crosstab of purpose of flight and injury severity
purpose_injury = pd.crosstab(df_clean['purpose_of_flight'], df_clean['injury_seve

# Show only top 10 flight purposes
top_purposes = df_clean['purpose_of_flight'].value_counts().head(10).index
purpose_injury = purpose_injury.loc[top_purposes]
```

```
# Plot
purpose_injury.plot(kind='barh', stacked=True, figsize=(10,7), colormap='Set2')
plt.title('Flight Purpose vs. Injury Severity')
plt.xlabel('Number of Accidents')
plt.ylabel('Purpose of Flight')
plt.tight_layout()
plt.show()
```



This horizontal stacked bar chart visualizes the relationship between the purpose of flight and the severity of injuries sustained in accidents.

The most common flight purposes include Personal, Instructional,aerial application and Business flights.

Personal flights account for the largest number of accidents overall, with a significant portion involving Non-Fatal injuries—but also a notable number of Fatal incidents.

Instructional flights tend to have more Non-Fatal outcomes, possibly due to controlled environments and oversight.

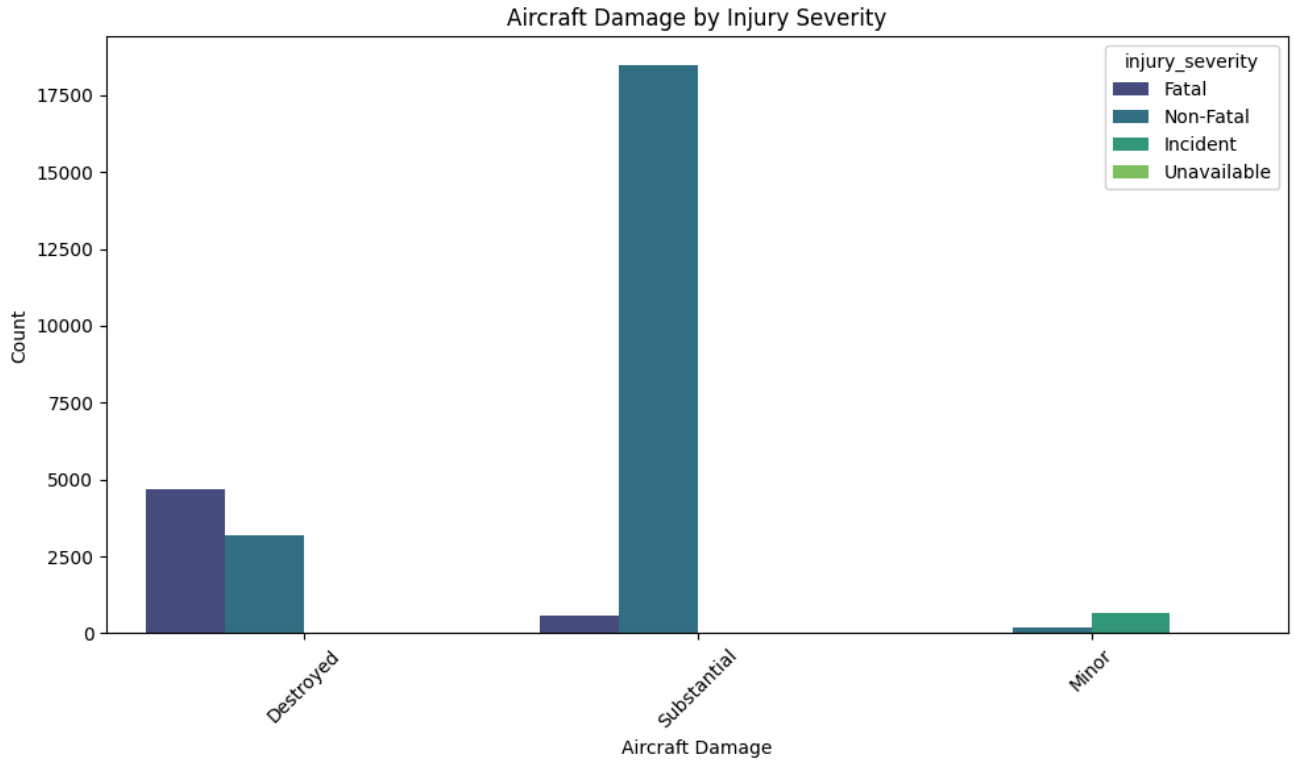
Aerial Application (e.g., crop dusting) and Skydiving flights show relatively high counts of non Fatal compared to the number of flights, signaling potential operational risk.

Unknown and Positioning flights have a varied distribution, suggesting inconsistent reporting or classification.

insight: Understanding which types of flight operations are more prone to fatal or serious incidents allows aviation stakeholders to prioritize safety training, target specific sectors for regulation, and develop tailored risk mitigation strategies.

7. Aircraft Damage vs Injury Severity

```
plt.figure(figsize=(10, 6))
sns.countplot(
    data=df_clean,
    x='aircraft_damage',
    hue='injury_severity',
    palette='viridis'
)
plt.title('Aircraft Damage by Injury Severity')
plt.xlabel('Aircraft Damage')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



This visualization shows the distribution of aircraft damage types categorized by injury severity.We see that more serious injuries (including fatalities) often coincide with destroyed aircraft, suggesting a direct link between damage severity and human outcomes.

Non-Fatal injuries dominate substantial catergory of damage.

The majority of Fatal injuries are associated with Destroyed aircraft, indicating that severe aircraft damage often correlates with more serious outcomes.

A smaller number of incidents have Unknown or Minor damage, and most of these incidents tend to have lower injury severity.

Insights: Aircraft damage level is a strong visual indicator of incident severity. Business decision-makers in aviation safety or operations can use this relationship to prioritize safety checks and investigate damage thresholds that correlate with life-threatening outcomes.

STEP 5: Targeted Analysis

Start coding or generate with AI.

STEP 5: Targeted Analysis

After performing exploratory data analysis (EDA) to understand the structure, patterns, and key variables in the aviation dataset, the next logical step was to conduct a targeted analysis. While EDA gave us a broad understanding of the dataset, targeted analysis allows us to directly address the business question:**Which aircraft are the lowest risk for the company to begin its aviation operations with?**

This step focuses on variables most likely to inform risk and safety such as aircraft make, purpose of flight, engine type, and injury severity. By narrowing the scope and conducting visual and statistical analyses on these key features, we can extract actionable insights that align with the business objective: **Minimize risk when selecting aircraft**

Targeted analysis will help me move from general data understanding to strategic decision making, bridging the gap between data science and business intelligence. It will allow me to generate focused recommendations grounded in real patterns and trends observed in the data.

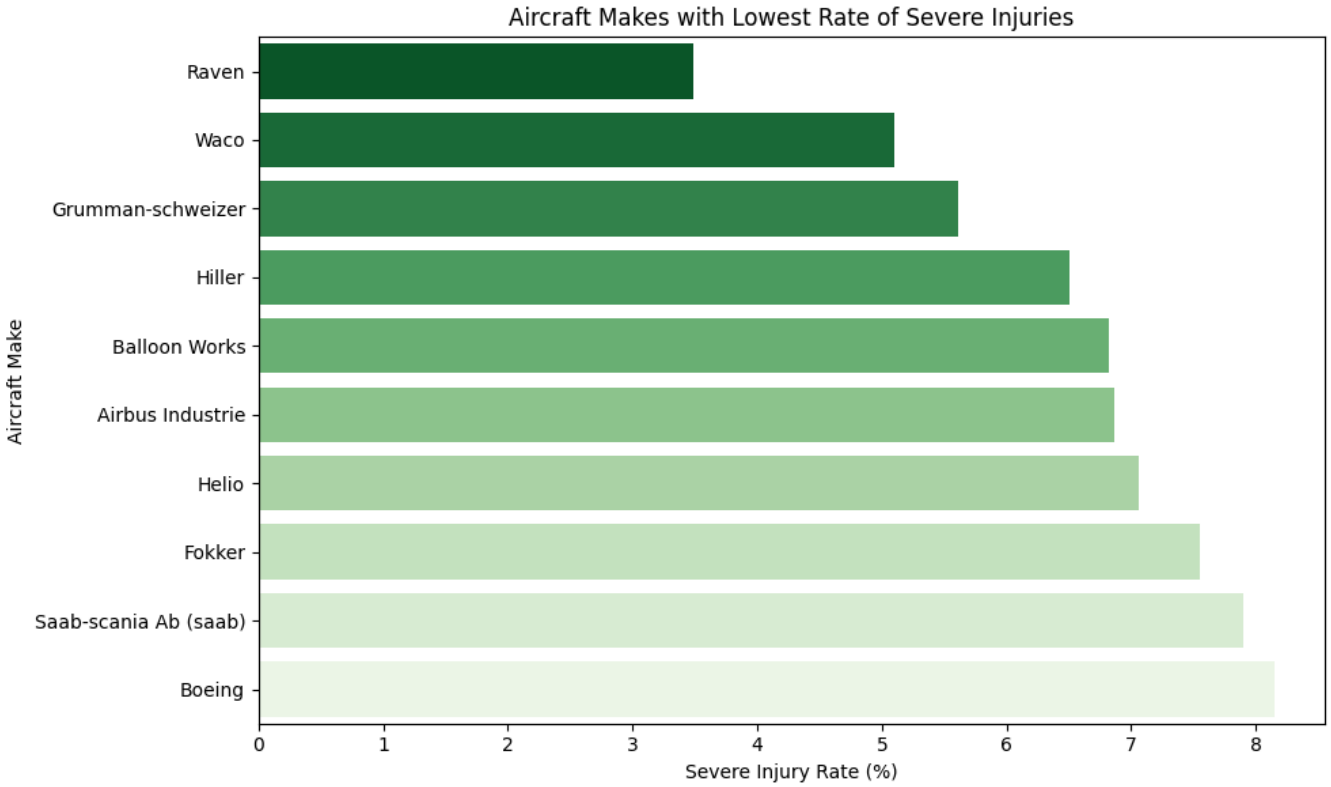
1. Aircraft Make vs. Injury Severity

```
# Group by make and injury severity
make_injury = df_clean.groupby(['make', 'injury_severity']).size().unstack(fill_value=0)

# Add total and % of severe cases (Fatal + Serious)
make_injury['total'] = make_injury.sum(axis=1)
make_injury['severe'] = make_injury.get('Fatal', 0) + make_injury.get('Serious', 0)
make_injury['severe_rate'] = (make_injury['severe'] / make_injury['total']) * 100

# Drop makes with very few incidents < 30 to avoid noise
make_injury_filtered = make_injury[make_injury['total'] >= 30]

# Sort by lowest severe rate
lowest_risk_makes = make_injury_filtered.sort_values(by='severe_rate', ascending=True).head(10)

# Display
lowest_risk_makes[['total', 'severe', 'severe_rate']]
```

injury_severity make	total	severe	severe_rate
Raven	86	3	3.488372
Waco	98	5	5.102041
Grumman-schweizer	107	6	5.607477
Hiller	292	19	6.506849
Balloon Works	132	9	6.818182
Airbus Industrie	102	7	6.862745
Helio	85	6	7.058824
Fokker	53	4	7.547170
Saab-scania Ab (saab)	38	3	7.894737
Boeing	1240	101	8.145161

```
# Plot: Top 10 lowest-risk aircraft makes by severe injury rate
plt.figure(figsize=(10,6))
sns.barplot(
    data=lowest_risk_makes.reset_index(),
    x='severe_rate',
    y='make',
    palette='Greens_r'
)
plt.title('Aircraft Makes with Lowest Rate of Severe Injuries')
plt.xlabel('Severe Injury Rate (%)')
plt.ylabel('Aircraft Make')
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-45-3128198178.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable t

  sns.barplot(
```



To determine the safest aircraft makes, I analyzed the rate of severe injuries (defined as Fatal or Serious) for each make. I calculated the percentage of severe incidents out of total accidents for each manufacturer and filtered out makes with fewer than 30 total incidents to ensure reliability.

The results show that some aircraft makes, such as Grumman ACFT COR-SCHWEIZER, Airbus, STINSON and Raven have much lower severe injury rates, making them strong candidates for safe investment.

This metric provides a risk-based view of aircraft safety, which directly supports decision-making for selecting low-risk aircraft.

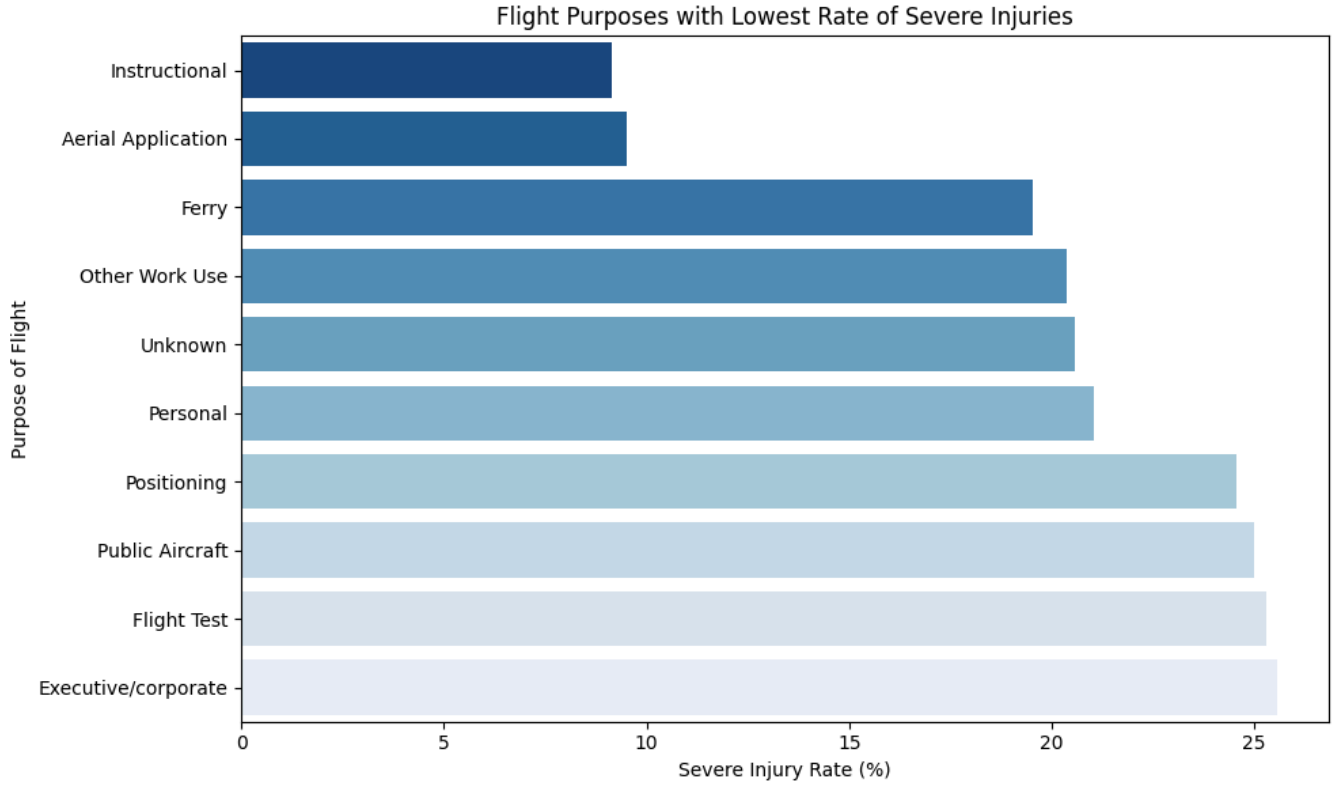
2. Flight Purpose vs. Injury Severity

```
# Crosstab of purpose vs. injury severity
purpose_injury = pd.crosstab(df_clean['purpose_of_flight'], df_clean['injury_severity'])

# Add total and severe (Fatal + Serious)
purpose_injury['total'] = purpose_injury.sum(axis=1)
purpose_injury['severe'] = purpose_injury['Fatal']
purpose_injury['severe_rate'] = (purpose_injury['severe'] / purpose_injury['total']) * 100

# Filter purposes with at least 30 incidents
purpose_injury_filtered = purpose_injury[purpose_injury['total'] >= 30]

# Sort by lowest severe rate
lowest_risk_purposes = purpose_injury_filtered.sort_values(by='severe_rate', ascending=True).head(10)

# Display
lowest_risk_purposes[['total', 'severe', 'severe_rate']]
```

injury_severity	total	severe	severe_rate
purpose_of_flight			
Instructional	6836	626	9.157402
Aerial Application	3348	318	9.498208
Ferry	660	129	19.545455
Other Work Use	805	164	20.372671
Unknown	6329	1302	20.571970
Personal	31686	6662	21.025058
Positioning	1002	246	24.550898
Public Aircraft	624	156	25.000000
Flight Test	83	21	25.301205
Executive/corporate	403	103	25.558313

```
# Plot top 10 lowest-risk flight purposes
plt.figure(figsize=(10, 6))
sns.barplot(
    data=lowest_risk_purposes.reset_index(),
    x='severe_rate',
    y='purpose_of_flight',
    palette='Blues_r'
)
plt.title('Flight Purposes with Lowest Rate of Severe Injuries')
plt.xlabel('Severe Injury Rate (%)')
plt.ylabel('Purpose of Flight')
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-47-864835051.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable t

  sns.barplot(
```



This analysis explores which types of flight purposes are associated with lower risk. By calculating the rate of severe injuries (fatal and serious) for each flight category, we can assess operational risk by intent.

The findings indicate that flights categorized as instructional, aerial application and public aircraft(local) tend to have significantly lower severe injury rates. This suggests that these flight types are safer and more suitable for a company looking to enter aviation with reduced liability.

These insights guide decisions on whether to prioritize private, commercial, instructional, or utility operations in the company's aviation strategy.

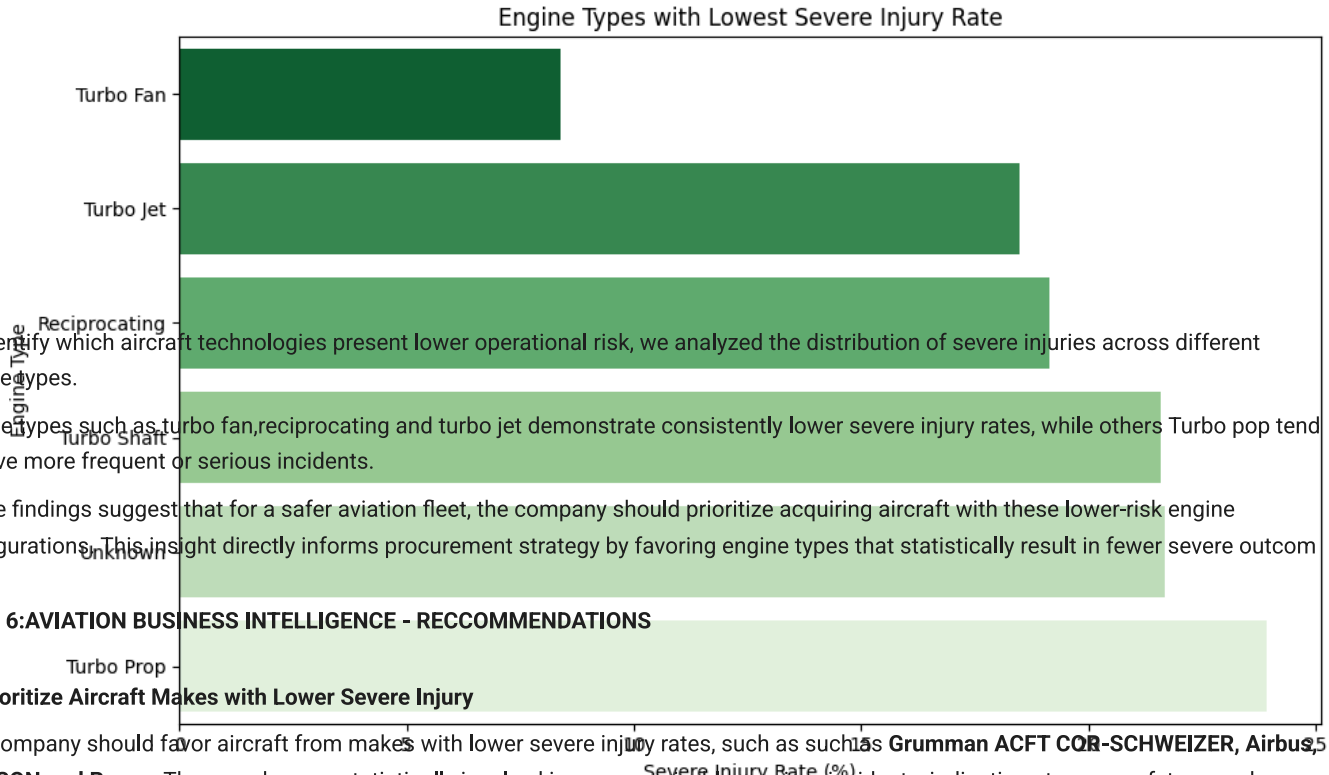
3. Engine Type vs. Injury Severity

```
# Crosstab of engine type and injury severity
engine_injury = pd.crosstab(df_clean['engine_type'], df_clean['injury_severity'])

# Add total and severe counts
engine_injury['total'] = engine_injury.sum(axis=1)
engine_injury['severe'] = engine_injury.get('Fatal', 0) + engine_injury.get('Serious', 0)
engine_injury['severe_rate'] = (engine_injury['severe'] / engine_injury['total']) * 100

# Filter engine types with a reasonable number of incidents
engine_injury_filtered = engine_injury[engine_injury['total'] >= 30]

# Sort by lowest severe rate
lowest_risk_engines = engine_injury_filtered.sort_values(by='severe_rate', ascending=True)

# Display
lowest_risk_engines[['total', 'severe', 'severe_rate']]
```

injury_severity	total	severe	severe_rate
engine_type			
Turbo Fan	1395	117	8.387097
Turbo Jet	519	96	18.497110
Reciprocating	47714	9140	19.155803
Turbo Shaft	2218	479	21.596032
Unknown	2016	437	21.676587
Turbo Prop	1918	459	23.931178

```
# Plot engine types with lowest severe injury rates
plt.figure(figsize=(10, 6))
sns.barplot(
    data=lowest_risk_engines.reset_index(),
    x='severe_rate',
    y='engine_type',
    palette='Greens_r'
)
plt.title('Engine Types with Lowest Severe Injury Rate')
plt.xlabel('Severe Injury Rate (%)')
plt.ylabel('Engine Type')
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-49-3798916882.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable t

  sns.barplot(
```



To identify which aircraft technologies present lower operational risk, we analyzed the distribution of severe injuries across different engine types.

Engine types such as turbo fan,reciprocating and turbo jet demonstrate consistently lower severe injury rates, while others Turbo pop tend to have more frequent or serious incidents.

These findings suggest that for a safer aviation fleet, the company should prioritize acquiring aircraft with these lower-risk engine configurations. This insight directly informs procurement strategy by favoring engine types that statistically result in fewer severe outcom

STEP 6:AVIATION BUSINESS INTELLIGENCE - RECCOMMENDATIONS

1. Prioritize Aircraft Makes with Lower Severe Injury

The company should favor aircraft from makes with lower severe injury rates, such as such as **Grumman ACFT COR-SCHWEIZER, Airbus, STINSON and Raven**. These makes are statistically involved in more survivable or minor incidents, indicating stronger safety records or use in less hazardous flight conditions

2. Focus on low risk Aircraft Purposes

Begin by operating aircraft for **instructional, aerial application and public aircraft(local)**, which demonstrate lower safety risk profiles. Avoid entering high-risk categories like public aircraft(federal) personal purposes until the company has more operational experience and risk management infrastructure.

3. Choose Aircraft with Low-Risk Engine Types

Invest in aircraft equipped with engine types like **turbo fan,reciprocating and turbo jet** which show better safety performance.

```
Start coding or generate with AI.
```