Machine Learning for Aviation Safety : Accident Trend Forecasting

In [1]:
```
#from google.colab import drive
#drive.mount('/content/drive')
```

1.0 **Business Understanding**

Aviation safety is paramount in the airline industry, where each accident can result in catastrophic consequences, not only in terms of human life but also reputational and financial losses. The National Transportation Safety Board (NTSB) has collected extensive accident data from 1962 to 2022, offering a unique opportunity to investigate patterns, root causes, and contributing factors of aviation accidents in the U.S.

By leveraging machine learning and data science, we aim to uncover insights and build predictive models to improve situational awareness, inform safety protocols, and ultimately reduce the frequency and severity of aviation incidents.

Problem Statement

Despite advances in aviation technology and regulation, accidents continue to occur due to a variety of factors such as weather, pilot error, aircraft condition, and flight phase. The challenge lies in understanding which conditions most often lead to severe outcomes and forecasting future risks based on historical trends.

Project Objectives

1. Descriptive Analysis:

- Explore trends in aviation accidents over the decades (frequency, location, severity).
- Identify the most common causes and contributing factors.

2. Predictive Modeling:

- Develop a time series model to forecast future accident trends.

3. Interpretability:

- Use feature importance to explain what drives severe accidents.

4. Recommendations:

- Suggest actionable insights for aviation safety regulators and flight operators.

Key Business Questions

1. What flight phases and weather conditions are most associated with fatal accidents?
2. Can we predict the severity of an accident based on pre-incident conditions?
3. Are accident rates increasing or decreasing over time?
4. Which types of aircraft or operations are most at risk?
5. How can predictive models assist in safety planning and incident prevention?

Metrics of Success

1. For Time Series:

- MAE / RMSE – Forecast error
- MAPE – Percentage error for trend forecasting
- Visual Fit – Actual vs. Predicted graph quality

2. Business Metrics:

- Actionable insights that can inform safety recommendations
- Ability to highlight the top 5 features contributing to high-severity accidents
- A model that can generalize across different types of flights and conditions

2.0 **Data Understanding**

1. Dataset Overview

- Records: 88,889
- Columns: 31
- Time Range: 1962–2022
- Source: NTSB Aviation Accident Database

2. Key Columns by Category

- Identifiers & Meta:

Event.Id, Investigation.Type, Accident.Number

3. Time & Location:

- Event.Date: Primary time series axis
- Location; Country, Latitude, Longitude

4. Aircraft & Operator Info:

- Make, Model, Aircraft.Category, Engine.Type
- Air.carrier, Operator

5. Incident Context

- Broad.phase.of.flight – Phase during which the accident occurred
- Weather.Condition – Visual/Instrument meteorological conditions
- Purpose.of.flight – Personal, Commercial, Instructional, etc.

6. Accident Impact

- Injury.Severity–Total.Fatal.Injuries, Total.Serious.Injuries, etc.
- Aircraft.damage – Minor, Substantial, Destroyed

Exploring the dataset

In [2]:
```
#Importing the necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
#Loading the dataset
data = pd.read_csv("AviationData.csv", encoding='latin1')
```

In [4]:
```
data
```

Out[4]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Co
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	U
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	U
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	U
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	U
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	U
...	...	...	...	...	...	
88884	20221227106491	Accident	ERA23LA093	2022-12-26	Annapolis, MD	U
88885	20221227106494	Accident	ERA23LA095	2022-12-26	Hampton, NH	U
88886	20221227106497	Accident	WPR23LA075	2022-12-26	Payson, AZ	U
88887	20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, UT	U
88888	20221230106513	Accident	ERA23LA097	2022-12-29	Athens, GA	U

88889 rows × 31 columns

In [5]:
```
data.shape
```

Out[5]:
```
(88889, 31)
```

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88889 entries, 0 to 88888
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Event.Id                88889 non-null  object 
 1   Investigation.Type      88889 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            50132 non-null  object 
 9   Airport.Name            52704 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     87507 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             81793 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           82505 non-null  object 
 30  Publication.Date        75118 non-null  object 
dtypes: float64(5), object(26)
memory usage: 21.0+ MB
```

In [7]:
```
data.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
Number.of.Engines	82805.0	1.146585	0.446510	0.0	1.0	1.0	1.0	8.0
Total.Fatal.Injuries	77488.0	0.647855	5.485960	0.0	0.0	0.0	0.0	349.0
Total.Serious.Injuries	76379.0	0.279881	1.544084	0.0	0.0	0.0	0.0	161.0
Total.Minor.Injuries	76956.0	0.357061	2.235625	0.0	0.0	0.0	0.0	380.0
Total.Uninjured	82977.0	5.325440	27.913634	0.0	0.0	1.0	2.0	699.0

In [8]:
```
data.describe(include='O').T
```

Out[8]:

	count	unique	top	freq
Event.Id	88889	87951	20001214X45071	3
Investigation.Type	88889	2	Accident	85015
Accident.Number	88889	88863	WPR23LA045	2
Event.Date	88889	14782	1982-05-16	25
Location	88837	27758	ANCHORAGE, AK	434
Country	88663	219	United States	82248
Latitude	34382	25592	332739N	19
Longitude	34373	27156	0112457W	24
Airport.Code	50132	10374	NONE	1488
Airport.Name	52704	24870	Private	240
Injury.Severity	87889	109	Non-Fatal	67357
Aircraft.damage	85695	4	Substantial	64148
Aircraft.Category	32287	15	Airplane	27617
Registration.Number	87507	79104	NONE	344
Make	88826	8237	Cessna	22227
Model	88797	12318	152	2367
Amateur.Built	88787	2	No	80312
Engine.Type	81793	12	Reciprocating	69530
FAR.Description	32023	31	091	18221
Schedule	12582	3	NSCH	4474
Purpose.of.flight	82697	26	Personal	49448
Air.carrier	16648	13590	Pilot	258
Weather.Condition	84397	4	VMC	77303
Broad.phase.of.flight	61724	12	Landing	15428
Report.Status	82505	17074	Probable Cause	61754
Publication.Date	75118	2924	25-09-2020	17019

In [9]:

```
data.isnull().sum()
```

Out[9]:

	0
Event.Id	0
Investigation.Type	0
Accident.Number	0
Event.Date	0
Location	52
Country	226
Latitude	54507
Longitude	54516
Airport.Code	38757
Airport.Name	36185
Injury.Severity	1000
Aircraft.damage	3194
Aircraft.Category	56602
Registration.Number	1382
Make	63
Model	92
Amateur.Built	102
Number.of.Engines	6084
Engine.Type	7096
FAR.Description	56866
Schedule	76307
Purpose.of.flight	6192
Air.carrier	72241
Total.Fatal.Injuries	11401
Total.Serious.Injuries	12510
Total.Minor.Injuries	11933
Total.Uninjured	5912
Weather.Condition	4492
Broad.phase.of.flight	27165
Report.Status	6384

	0
Publication.Date	13771

dtype: int64

In [10]:
```
data.duplicated().sum()
```

Out[10]: np.int64(0)

3.0 Data Preparation

Data Cleaning

After looking into my dataset and reviewing what categorical and numerical data I will require for my analysis, I will start by cleaning the dataset by:

1. Dropping columns that are unnecessary for my analysis
2. Dropping missing values
3. Rectifying column arrangement for uniformity
4. Checking for outliers and eliminating them

In [11]:
```
#Creating a new copy for analysis
df = data.copy(deep=True)
#Confirming the changes
df.head()
```

Out[11]:

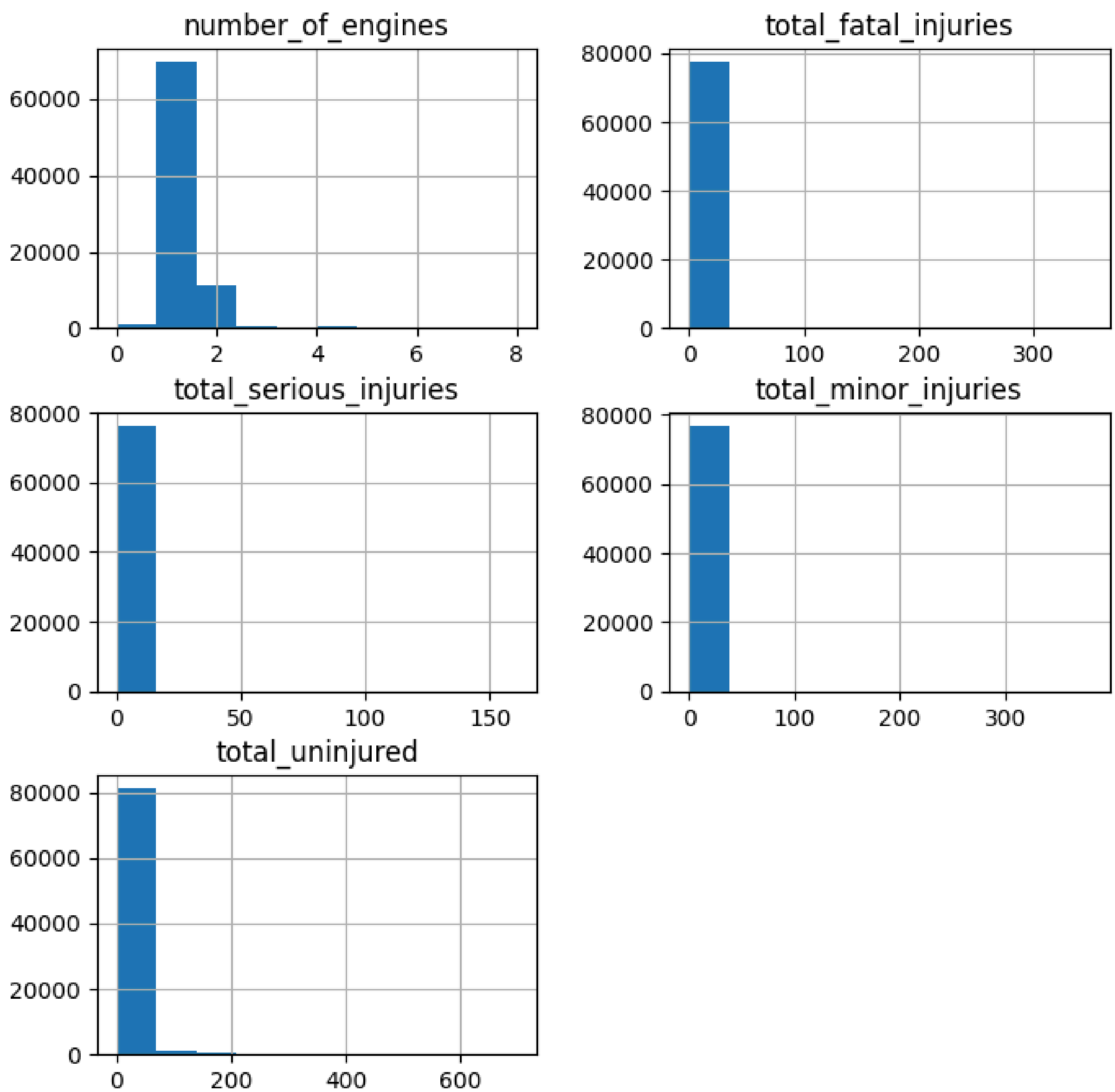
	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Countr
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	Unitec State
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	Unitec State
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	Unitec State
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	Unitec State
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	Unitec State

5 rows × 31 columns

In [12]:
```
#Dropping uneccessary columns
df.drop(columns=['Event.Id','Accident.Number','Latitude','Longitude','Airport.Code'
                  'Schedule','Air.carrier','Report.Status','Publication.Date','FAR.
                 'Registration.Number'],inplace= True,axis=1)
```

In [13]:
```
#Changing format for the columns
df.columns=df.columns.str.replace('.','_')

#Changing the columns to lower case
df.columns=df.columns.str.lower()
```

In [14]:
```
#Handling missing values for numeric columns
#Start by checking the frequency distribution for each numeric columns
df.hist(figsize=(8,8),bins=10);
```



In [15]:
```
#Handle the missing values using median
for column in df.select_dtypes(include=['number']).columns:
  df[column].fillna(df[column].median(),inplace=True)
```

In [16]:
```
#Handle the missing values using mode for categorical values
for column in df.select_dtypes(include=['object']).columns:
  df[column].fillna(df[column].mode()[0],inplace=True)
```

In [17]:
```
df.isnull().sum()
```

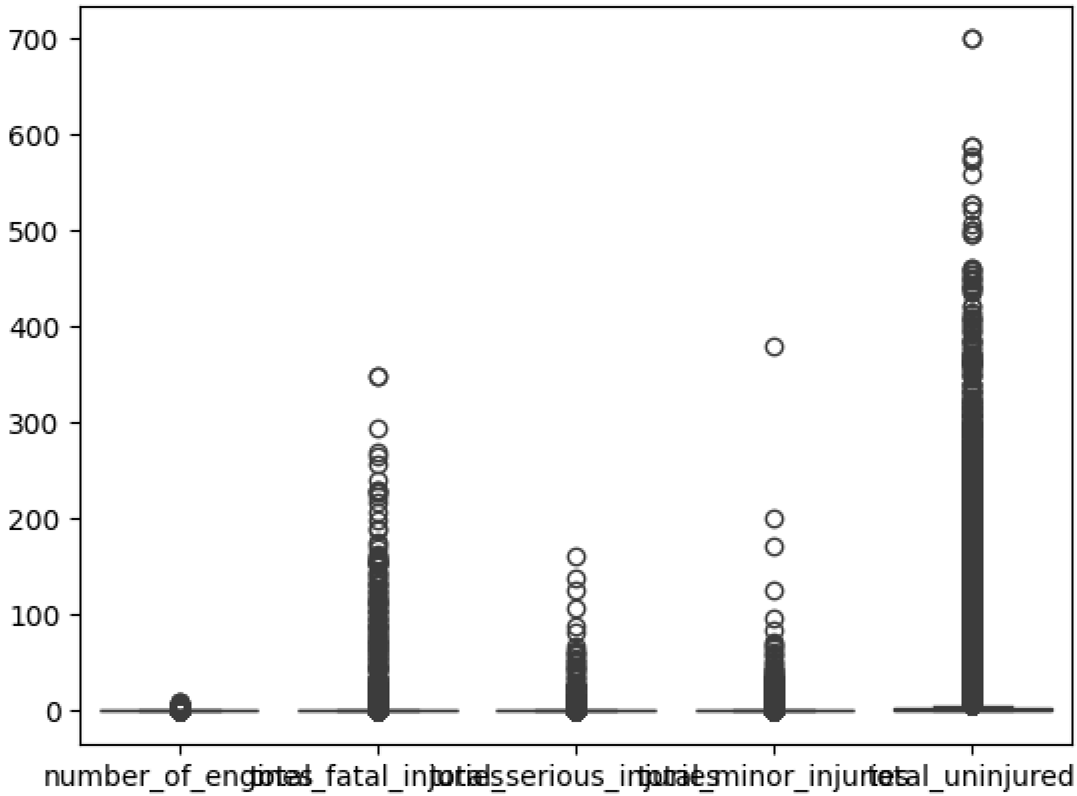
Out[17]:

	0
investigation_type	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

dtype: int64

In [18]:
```
#Checking for outliers
sns.boxplot(df)
```

Out[18]:
```
<Axes: >
```



Understanding and exploring each column

In [19]:
```
for col in df.columns:
    print(f"\n--- {col} ---")
    print(df[col].value_counts(dropna=False).head(10))
```

```
--- investigation_type ---
investigation_type
Accident    85015
Incident     3874
Name: count, dtype: int64

--- event_date ---
event_date
1982-05-16    25
1984-06-30    25
2000-07-08    25
1983-08-05    24
1984-08-25    24
1983-06-05    24
1986-05-17    24
2001-07-21    23
1982-10-03    23
1982-07-09    23
Name: count, dtype: int64

--- location ---
location
ANCHORAGE, AK      486
MIAMI, FL          200
ALBUQUERQUE, NM    196
HOUSTON, TX        193
CHICAGO, IL        184
FAIRBANKS, AK      174
TUCSON, AZ         142
ORLANDO, FL        132
PHOENIX, AZ        132
ENGLEWOOD, CO      131
Name: count, dtype: int64

--- country ---
country
United States     82474
Brazil              374
Canada              359
Mexico              358
United Kingdom      344
Australia           300
France              236
Spain               226
Bahamas             216
Germany             215
Name: count, dtype: int64

--- injury_severity ---
injury_severity
Non-Fatal    68357
Fatal(1)      6167
Fatal         5262
Fatal(2)      3711
Incident      2219
Fatal(3)      1147
```

```
Fatal(4)       812
Fatal(5)       235
Minor          218
Serious        173
Name: count, dtype: int64

--- aircraft_damage ---
aircraft_damage
Substantial    67342
Destroyed      18623
Minor           2805
Unknown          119
Name: count, dtype: int64

--- aircraft_category ---
aircraft_category
Airplane             84219
Helicopter            3440
Glider                 508
Balloon                231
Gyrocraft              173
Weight-Shift           161
Powered Parachute       91
Ultralight              30
Unknown                 14
WSFT                     9
Name: count, dtype: int64

--- make ---
make
Cessna     22290
Piper      12029
CESSNA      4922
Beech       4330
PIPER       2841
Bell        2134
Boeing      1594
BOEING      1151
Grumman     1094
Mooney      1092
Name: count, dtype: int64

--- model ---
model
152          2459
172          1756
172N         1164
PA-28-140     932
150           829
172M          798
172P          689
182           659
180           622
150M          585
Name: count, dtype: int64
```

```
--- amateur_built ---
amateur_built
No     80414
Yes     8475
Name: count, dtype: int64

--- number_of_engines ---
number_of_engines
1.0    75666
2.0    11079
0.0     1226
3.0      483
4.0      431
8.0        3
6.0        1
Name: count, dtype: int64

--- engine_type ---
engine_type
Reciprocating      76626
Turbo Shaft         3609
Turbo Prop          3391
Turbo Fan           2481
Unknown             2051
Turbo Jet            703
Geared Turbofan       12
Electric              10
NONE                   2
LR                     2
Name: count, dtype: int64

--- purpose_of_flight ---
purpose_of_flight
Personal              55640
Instructional         10601
Unknown                6802
Aerial Application     4712
Business               4018
Positioning            1646
Other Work Use         1264
Ferry                   812
Aerial Observation      794
Public Aircraft         720
Name: count, dtype: int64

--- total_fatal_injuries ---
total_fatal_injuries
0.0     71076
1.0      8883
2.0      5173
3.0      1589
4.0      1103
5.0       346
6.0       216
7.0       101
8.0        70
```

```
10.0       45
Name: count, dtype: int64

--- total_serious_injuries ---
total_serious_injuries
0.0     75799
1.0      9125
2.0      2815
3.0       629
4.0       258
5.0        78
6.0        41
7.0        27
9.0        16
10.0       13
Name: count, dtype: int64

--- total_minor_injuries ---
total_minor_injuries
0.0    73387
1.0    10320
2.0     3576
3.0      784
4.0      372
5.0      129
6.0       67
7.0       59
9.0       22
8.0       20
Name: count, dtype: int64

--- total_uninjured ---
total_uninjured
1.0    31013
0.0    29879
2.0    15988
3.0     4313
4.0     2662
5.0      887
6.0      500
7.0      281
8.0      163
9.0      128
Name: count, dtype: int64

--- weather_condition ---
weather_condition
VMC    81795
IMC     5976
UNK      856
Unk      262
Name: count, dtype: int64

--- broad_phase_of_flight ---
broad_phase_of_flight
Landing        42593
```

```
Takeoff        12493
Cruise         10269
Maneuvering     8144
Approach        6546
Climb           2034
Taxi            1958
Descent         1887
Go-around       1353
Standing         945
Name: count, dtype: int64
```

In [20]:
```
# Standardize the Injury.Severity column
df["injury_severity"] = df["injury_severity"].replace(
    to_replace=r"^Fatal(\(\d+\))?$",  # Matches "Fatal", "Fatal(1)", "Fatal(2)", et
    value="Fatal",
    regex=True
)
```

In [21]:
```
# Convert to uppercase and standardize known values
df["weather_condition"] = df["weather_condition"].str.upper()
```

In [22]:
```
# Convert to lowercase for make
df["make"] = df["make"].str.lower()
```

In [23]:
```
#Saving the cleaned dataset
df.to_csv('Clean Aviation.csv')
```

Feature Engineering

In [24]:
```
# Step 1: Normalize to uppercase
df["weather_condition"] = df["weather_condition"].str.upper()

# Step 2: Map weather codes to categories
weather_mapping = {
    "VMC": "Favorable",   # Visual Meteorological Conditions
    "IMC": "Challenging"  # Instrument Meteorological Conditions
}

df["weather_category"] = df["weather_condition"].map(weather_mapping)

# Step 3: Drop rows with unknown/missing weather conditions
df = df[df["weather_category"].notna()]

# Step 4: Check result
print(df["weather_category"].value_counts())
```

```
weather_category
Favorable      81795
Challenging     5976
Name: count, dtype: int64
```

In [25]:
```
#Distributing date-times
df['event_date'] = pd.to_datetime(df['event_date'])
df['year'] = df['event_date'].dt.year
```

```
df['month'] = df['event_date'].dt.month
df['day_of_week'] = df['event_date'].dt.dayofweek  # 0=Monday
df['quarter'] = df['event_date'].dt.quarter
```

In [26]:
```
# Combining injuries
df['total_injuries']=(df['total_minor_injuries']+df['total_fatal_injuries']
                      +df['total_serious_injuries'])
```

In [27]:
```
#Categorizing the fatalities
def categorize_fatalities(x):
    if pd.isna(x):
        return "Unknown"
    elif x == 0:
        return "None"
    elif x == 1:
        return "Single Fatality"
    elif 2 <= x <= 4:
        return "Few Fatalities"
    elif 5 <= x <= 9:
        return "Moderate Fatalities"
    else:
        return "Mass Fatality"

df["fatality_category"] = df["total_fatal_injuries"].apply(categorize_fatalities)

# View distribution
print(df["fatality_category"].value_counts())
```

```
fatality_category
None                   70528
Single Fatality         8673
Few Fatalities          7579
Moderate Fatalities      723
Mass Fatality            268
Name: count, dtype: int64
```

In [28]:
```
def categorize_total_injuries(x):
    if pd.isna(x):
        return "Unknown"
    elif x == 0:
        return "No Injuries"
    elif x == 1:
        return "Isolated Injury"
    elif 2 <= x <= 4:
        return "Few Injuries"
    elif 5 <= x <= 9:
        return "Moderate Injuries"
    else:
        return "Mass Casualties"

df["injury_severity_category"] = df["total_injuries"].apply(categorize_total_injuri

# View distribution
print(df["injury_severity_category"].value_counts())
```

```
injury_severity_category
No Injuries           48019
Isolated Injury       20322
Few Injuries          17412
Moderate Injuries      1499
Mass Casualties         519
Name: count, dtype: int64
```

In [29]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 87771 entries, 2 to 88888
Data columns (total 27 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   investigation_type        87771 non-null  object
 1   event_date                87771 non-null  datetime64[ns]
 2   location                  87771 non-null  object
 3   country                   87771 non-null  object
 4   injury_severity           87771 non-null  object
 5   aircraft_damage           87771 non-null  object
 6   aircraft_category         87771 non-null  object
 7   make                      87771 non-null  object
 8   model                     87771 non-null  object
 9   amateur_built             87771 non-null  object
 10  number_of_engines         87771 non-null  float64
 11  engine_type               87771 non-null  object
 12  purpose_of_flight         87771 non-null  object
 13  total_fatal_injuries      87771 non-null  float64
 14  total_serious_injuries    87771 non-null  float64
 15  total_minor_injuries      87771 non-null  float64
 16  total_uninjured           87771 non-null  float64
 17  weather_condition         87771 non-null  object
 18  broad_phase_of_flight     87771 non-null  object
 19  weather_category          87771 non-null  object
 20  year                      87771 non-null  int32
 21  month                     87771 non-null  int32
 22  day_of_week               87771 non-null  int32
 23  quarter                   87771 non-null  int32
 24  total_injuries            87771 non-null  float64
 25  fatality_category         87771 non-null  object
 26  injury_severity_category  87771 non-null  object
dtypes: datetime64[ns](1), float64(6), int32(4), object(16)
memory usage: 17.4+ MB
```

In [30]:
```
df['injury_severity'].value_counts()
```

Out[30]:

injury_severity	count
Non-Fatal	67937
Fatal	17255
Incident	2119
Minor	218
Serious	172
Unavailable	70

dtype: int64

In [31]:
```
df.isnull().sum()
```

Out[31]:

	0
investigation_type	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
weather_category	0
year	0
month	0
day_of_week	0
quarter	0
total_injuries	0
fatality_category	0
injury_severity_category	0

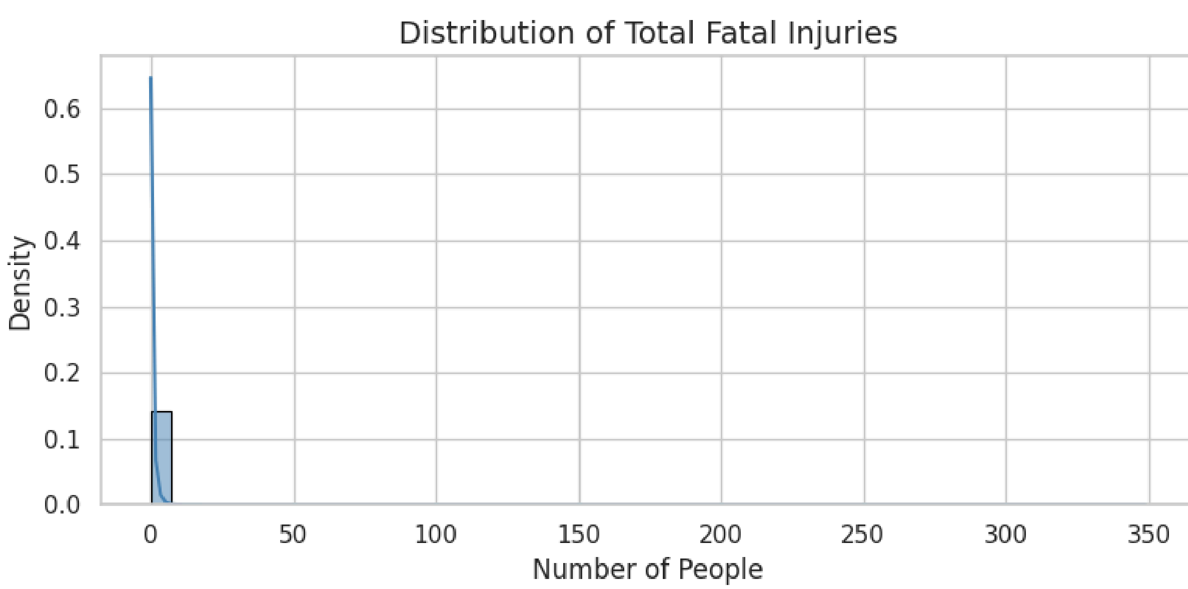
dtype: int64

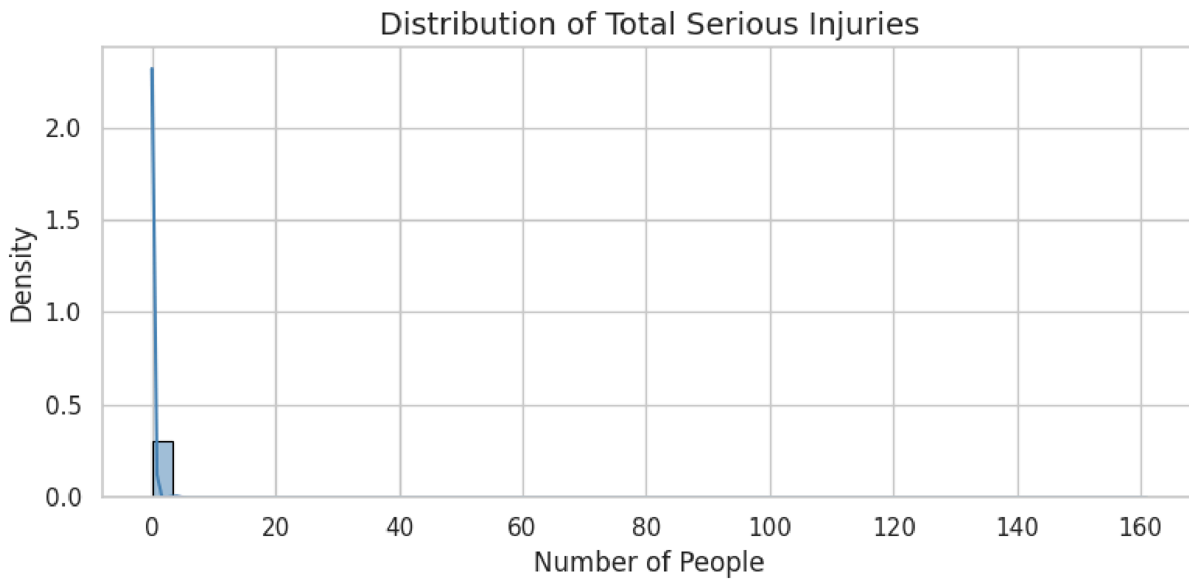
Exploratory Data Analysis (EDA)

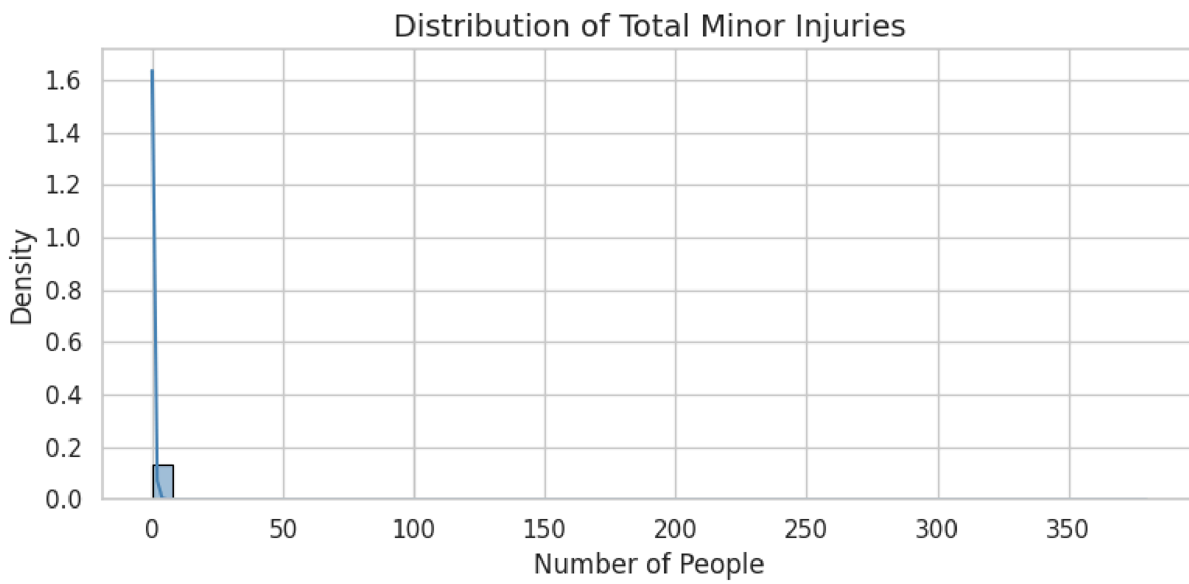
Univariate Analysis

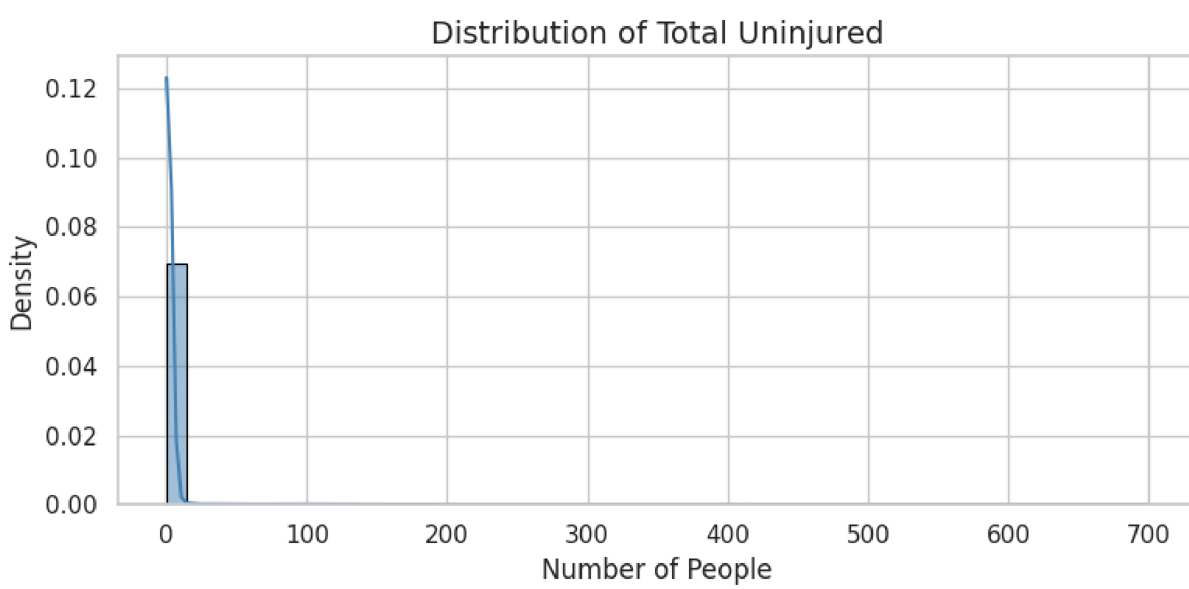
Numeric category

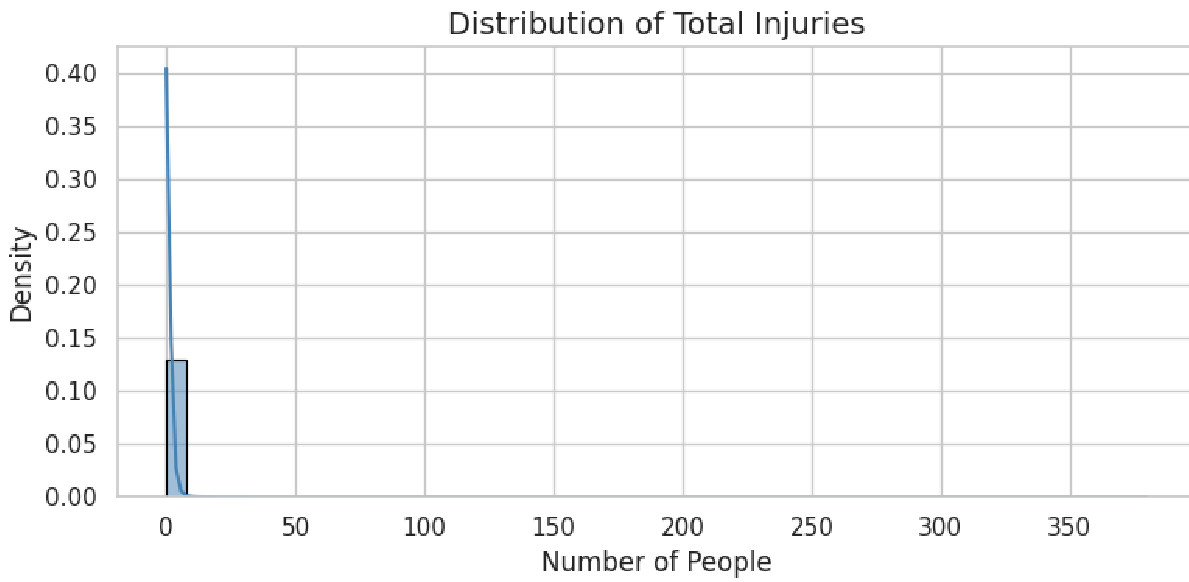
In [32]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# List of individual injury-related columns
injury_columns = [
    "total_fatal_injuries",
    "total_serious_injuries",
    "total_minor_injuries",
    "total_uninjured",
    "total_injuries"
]

# Set style
sns.set(style="whitegrid")

# Loop: one figure per injury column
for col in injury_columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[col].fillna(0), bins=50, kde=True, stat="density", color="steel
    plt.title(f"Distribution of {col.replace('_', ' ').title()}", fontsize=14)
    plt.xlabel("Number of People", fontsize=12)
    plt.ylabel("Density", fontsize=12)
    plt.tight_layout()
    plt.show()
```




Distribution of Total Serious Injuries
Density
Number of People
0.0
0.5
1.0
1.5
2.0
0
20
40
60
80
100
120
140
160


Distribution of Total Minor Injuries
Density
Number of People
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
50
100
150
200
250
300
350


Distribution of Total Uninjured
Density
Number of People
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
100
200
300
400
500
600
700



Observations:

- Extremely Skewed Distribution: The most striking observation is that the distribution is heavily skewed to the left (or positively skewed). The vast majority of the data points are concentrated very close to zero on the "Number of People" axis.

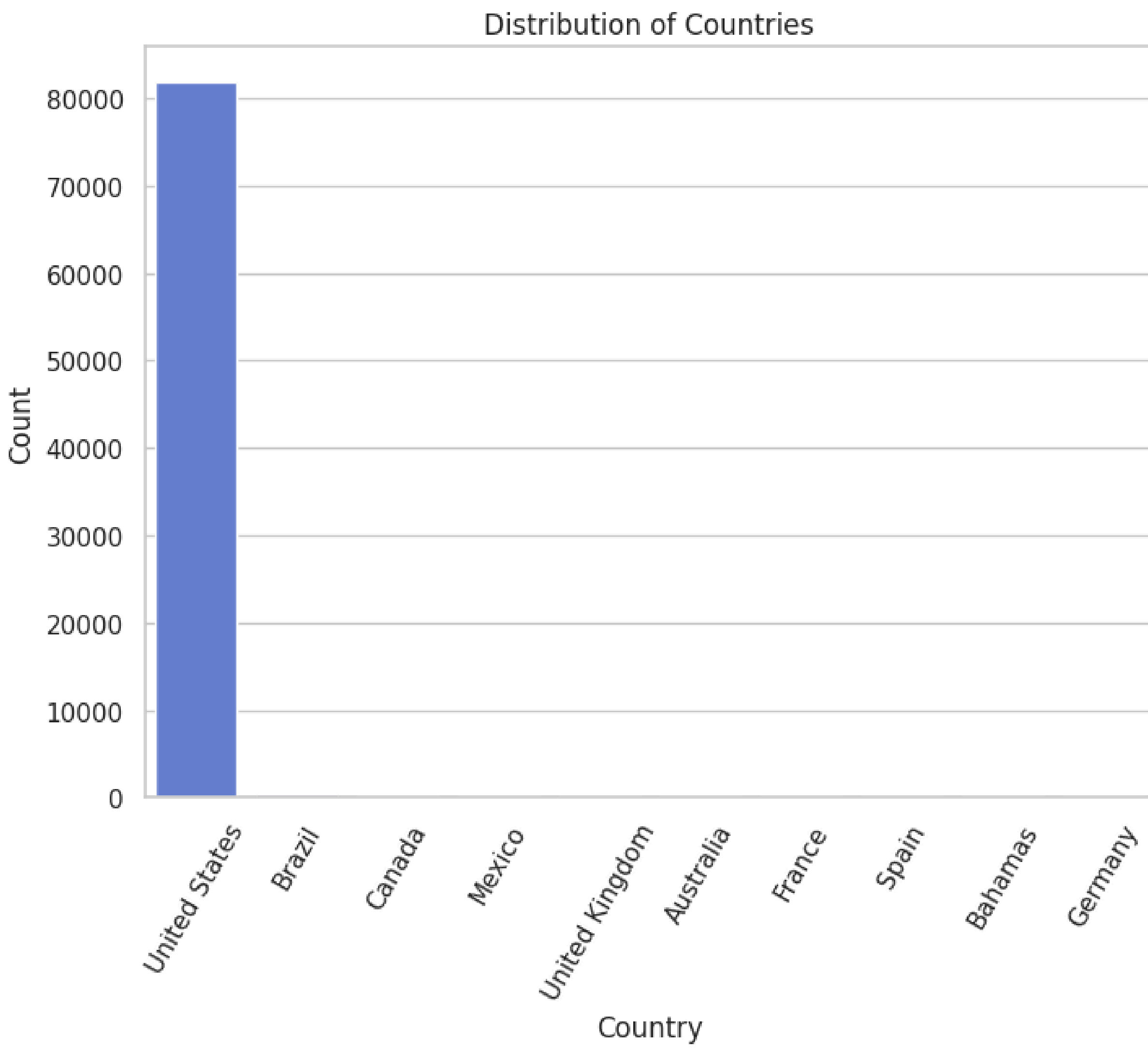
- Peak at Zero: There is a very tall bar (histogram bin) at or very near zero on the x-axis, indicating that a significant number of instances have 0 uninjured people.

Rapid Decline: The density (or frequency) drops off extremely rapidly as the "Number of People" increases from zero.

- Long Tail: Although the density is very low, there's a long tail extending out to potentially 600 or 700 on the "Number of People" axis, suggesting that there are a few instances with a very high number of uninjured people, but these are rare.

Categorical Data

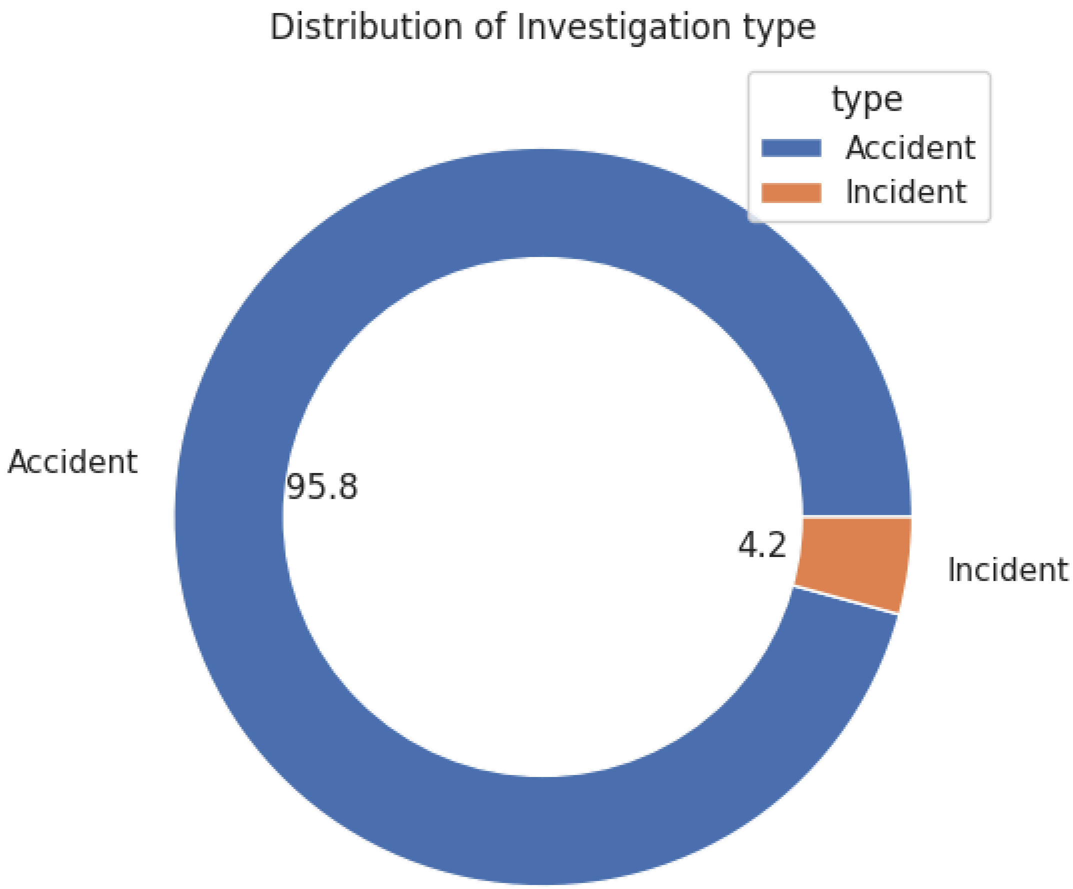
In [33]:
```
# Distribution of accidents per country
countries_count = df['country'].value_counts().head(10)
plt.figure(figsize=(8,6))
sns.barplot(x=countries_count.index,y=countries_count.values,palette='coolwarm')
plt.title('Distribution of Countries')
plt.xlabel('Country')
plt.ylabel('Count')
plt.xticks(rotation=60)
plt.show()
```



Observations:

The vast majority of accidents in the dataset occurred in the United States. This is expected as the NTSB (National Transportation Safety Board) primarily investigates accidents within the United States.

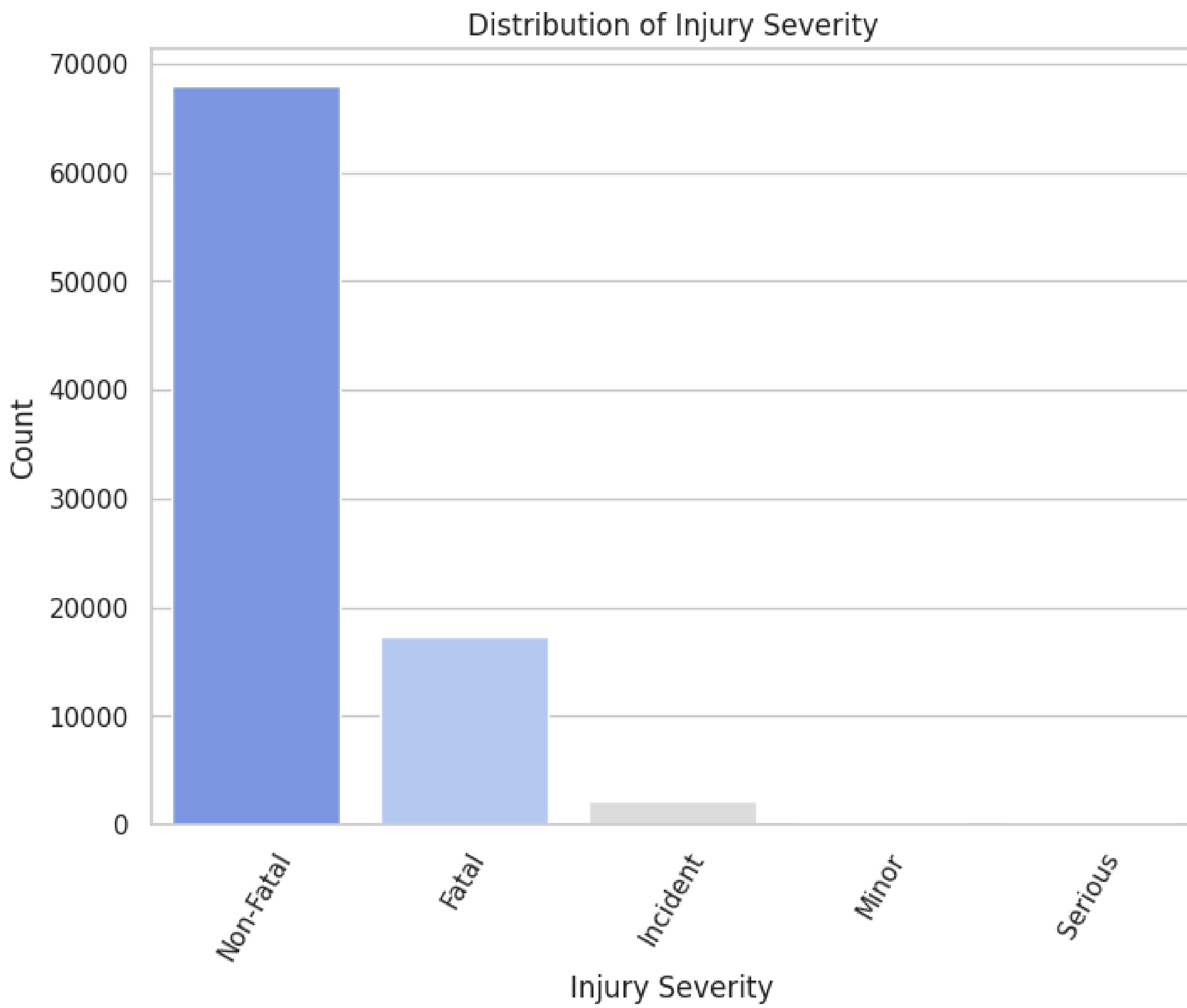
In [34]:
```
# Distribution of accidents per country
investigation_count = df['investigation_type'].value_counts().head(10)
plt.figure(figsize=(10,6))
plt.pie(investigation_count,labels=investigation_count.index,autopct='%1.1f')

#Create blank circle
centre_circle =plt.Circle((0,0),0.70,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

#Customize the plot
plt.title('Distribution of Investigation type')
plt.legend(title='type',loc ='upper right')
plt.show()
```



Observations:

Most investigations are labeled as "Accident" which is 96%.

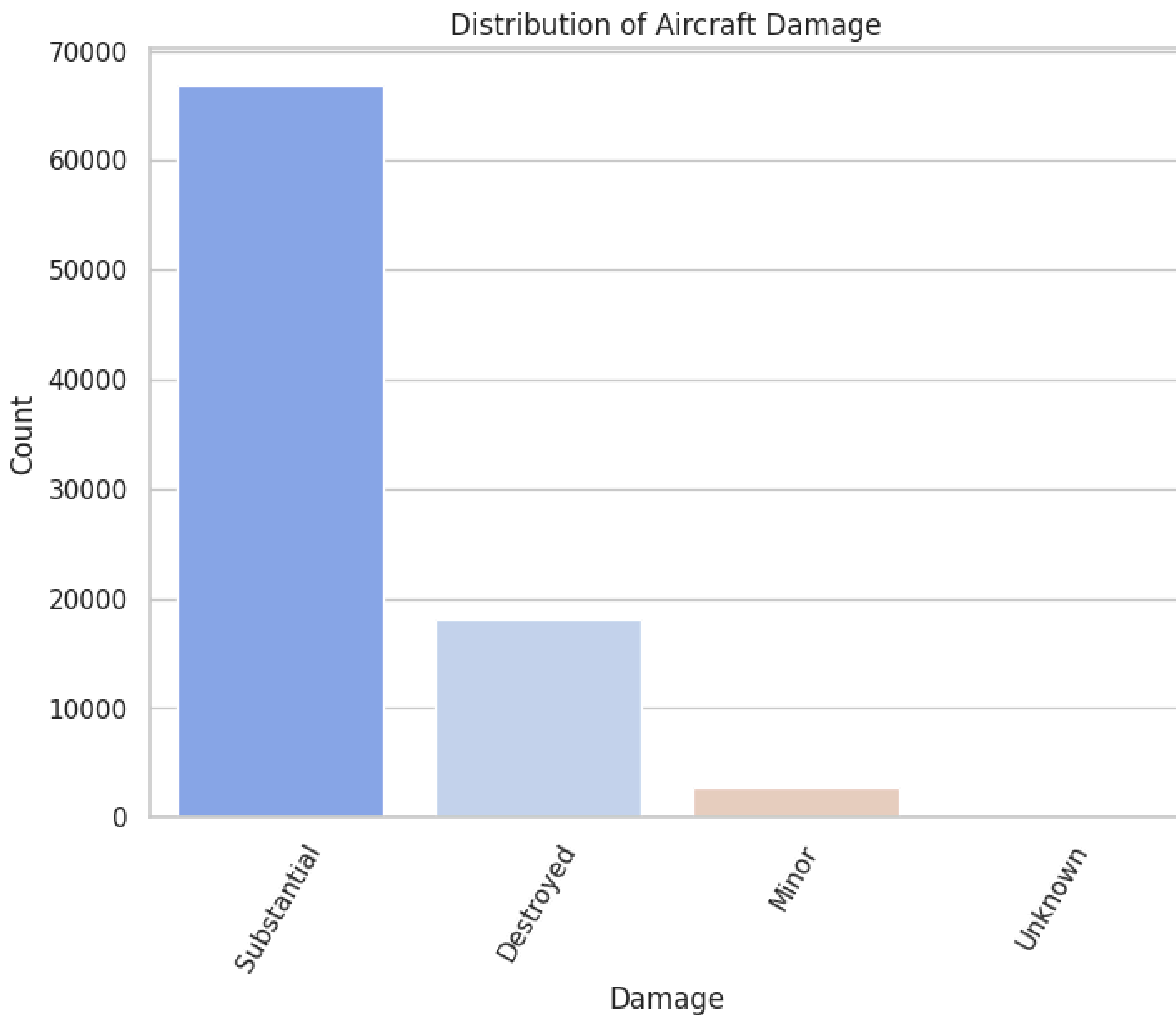
In [35]:
```
# Distribution of Injury severity
severity_count = df['injury_severity'].value_counts().head(5)
plt.figure(figsize=(8,6))
sns.barplot(x=severity_count.index,y=severity_count.values,palette='coolwarm')
plt.title('Distribution of Injury Severity')
plt.xlabel('Injury Severity')
plt.ylabel('Count')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- The most frequent outcome in terms of injury severity is "Non- Fatal" and "Fatal" with over a thousand cases.
- The other severities occur less frequently, with "Serious" being the least common among the displayed top 5.

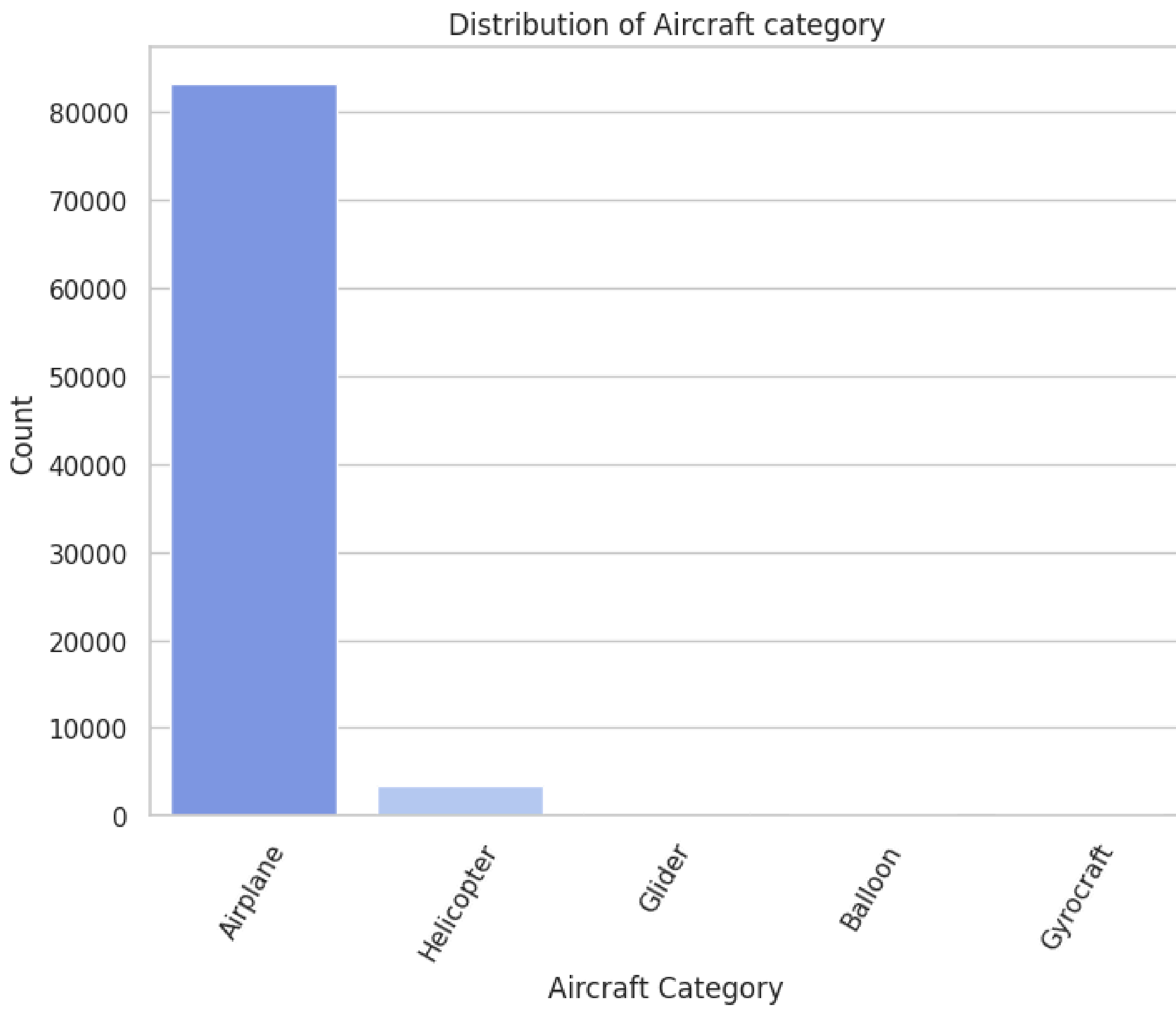
In [36]:
```
# Distribution of Aircraft Damage
damage_count = df['aircraft_damage'].value_counts().head(5)
plt.figure(figsize=(8,6))
sns.barplot(x=damage_count.index,y=damage_count.values,palette='coolwarm')
plt.title('Distribution of Aircraft Damage')
plt.xlabel('Damage')
plt.ylabel('Count')
plt.xticks(rotation=60)
plt.show()
```



Observations

A large portion of the accidents resulted in "Substantial" damage to the aircraft, followed by "Destroyed". "Minor" damage is the least common.

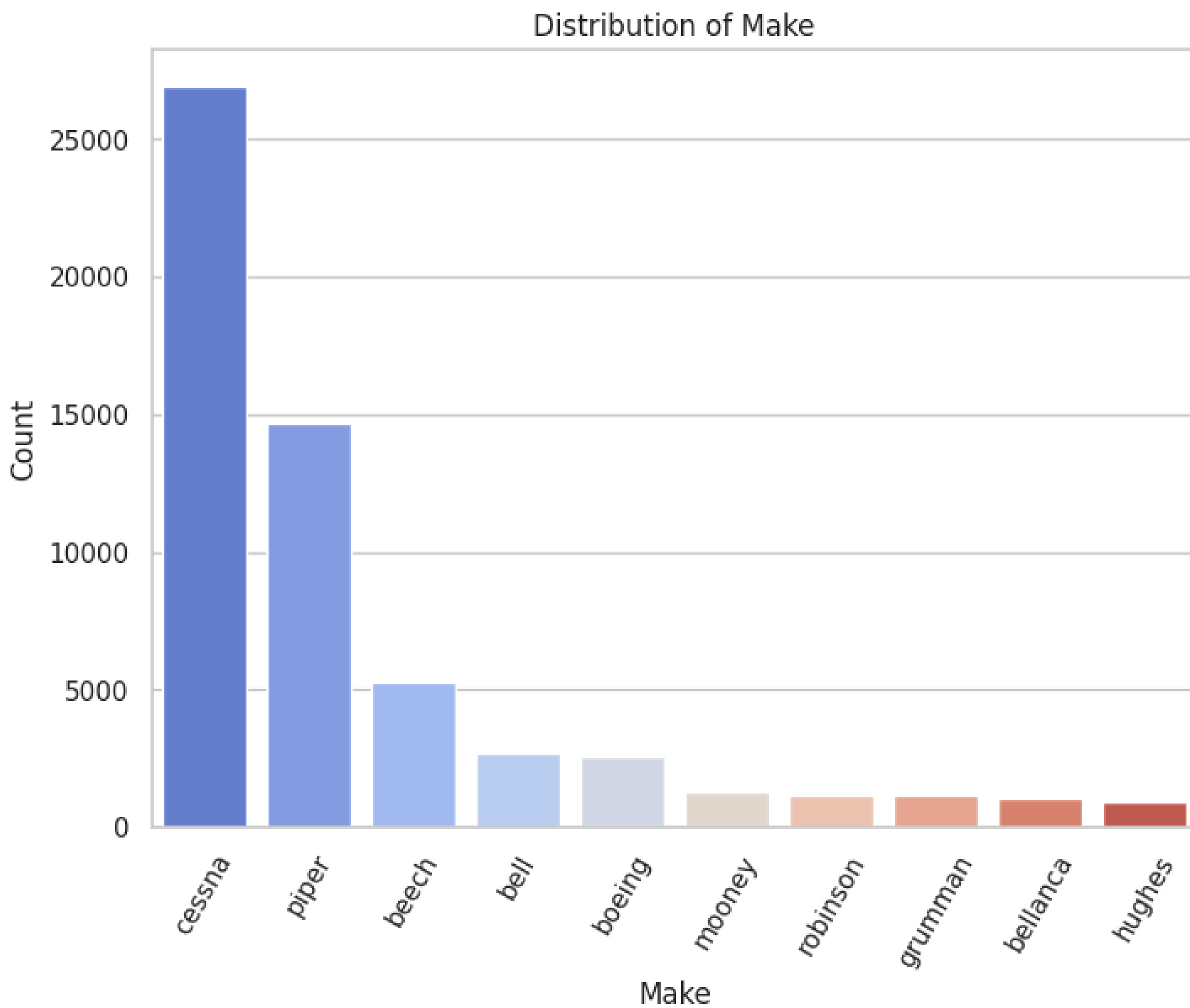
In [37]:
```
# Distribution of Aircraft Category
aircraft_count = df['aircraft_category'].value_counts().head(5)
plt.figure(figsize=(8,6))
sns.barplot(x=aircraft_count.index,y=aircraft_count.values,palette='coolwarm')
plt.title('Distribution of Aircraft category')
plt.xlabel('Aircraft Category')
plt.ylabel('Count')
plt.xticks(rotation=60)
plt.show()
```



Observations

"Airplane" is by far the most common aircraft category involved in accidents, significantly more frequent than "Helicopter", "Glider", or other types.

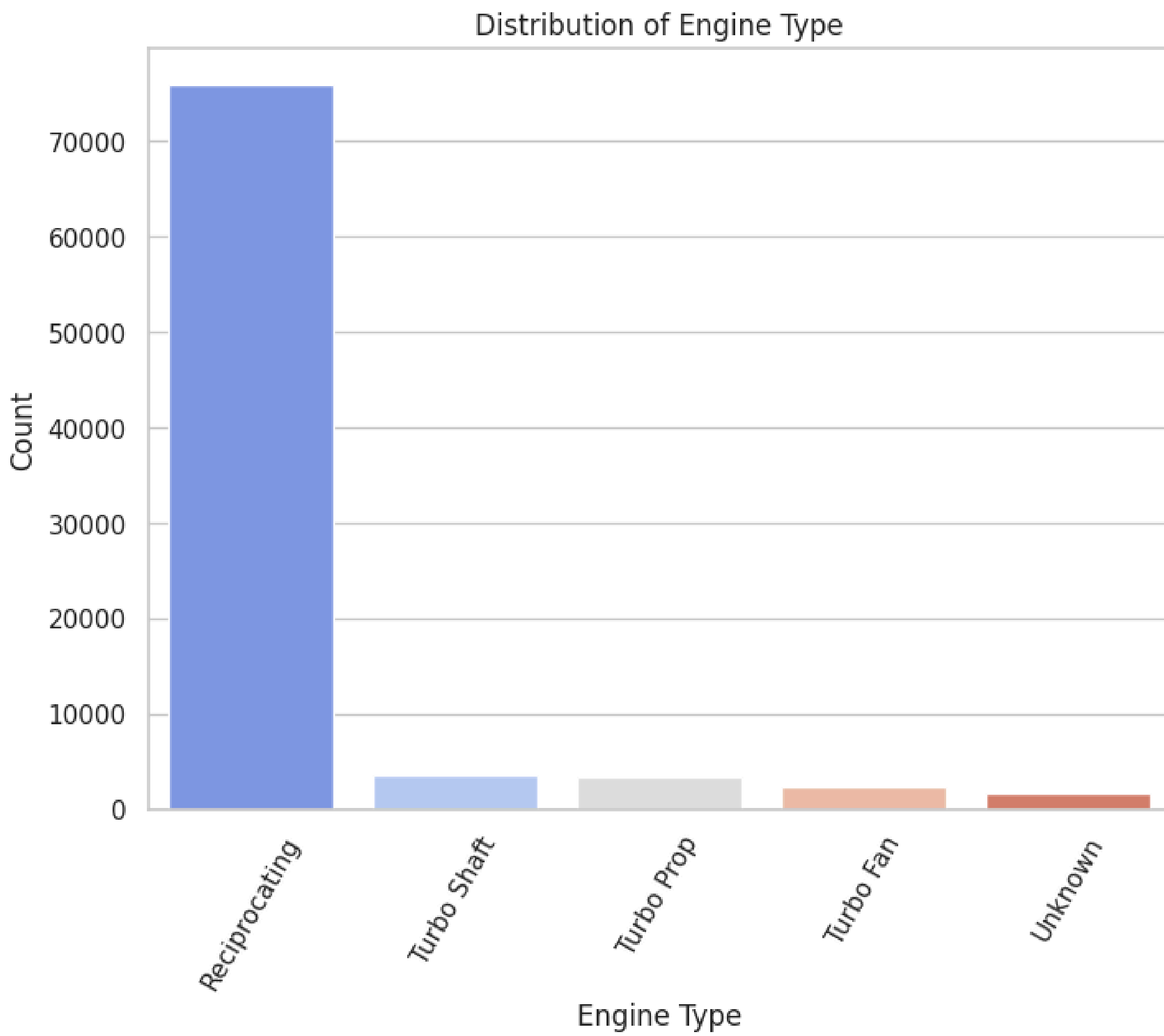
In [38]:
```
# Distribution of Make
make_count = df['make'].value_counts().head(10)
plt.figure(figsize=(8,6))
sns.barplot(x=make_count.index,y=make_count.values,palette='coolwarm')
plt.title('Distribution of Make')
plt.xlabel('Make')
plt.ylabel('Count')
plt.xticks(rotation=60)
plt.show()
```



Observations

Cessna is the most frequent make of aircraft involved in accidents, followed by Piper each recording more than 10000 cases.

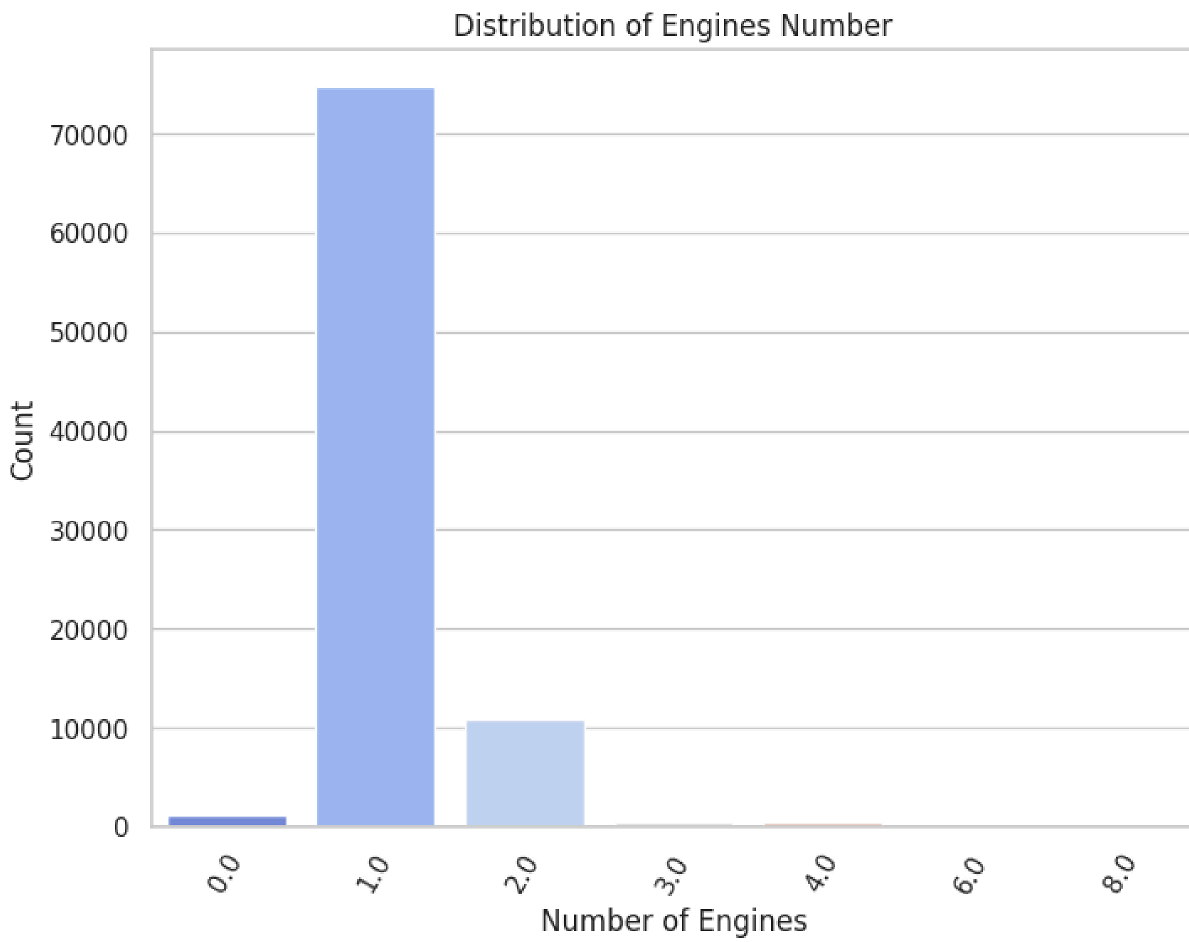
In [39]:
```
# Distribution of Engine type
engine_count = df['engine_type'].value_counts().head(5)
plt.figure(figsize=(8,6))
sns.barplot(x=engine_count.index,y=engine_count.values,palette='coolwarm')
plt.title('Distribution of Engine Type')
plt.xlabel('Engine Type')
plt.ylabel('Count')
plt.xticks(rotation=60)
plt.show()
```



Observations

Reciprocating" engines are involved in the highest number of accidents.

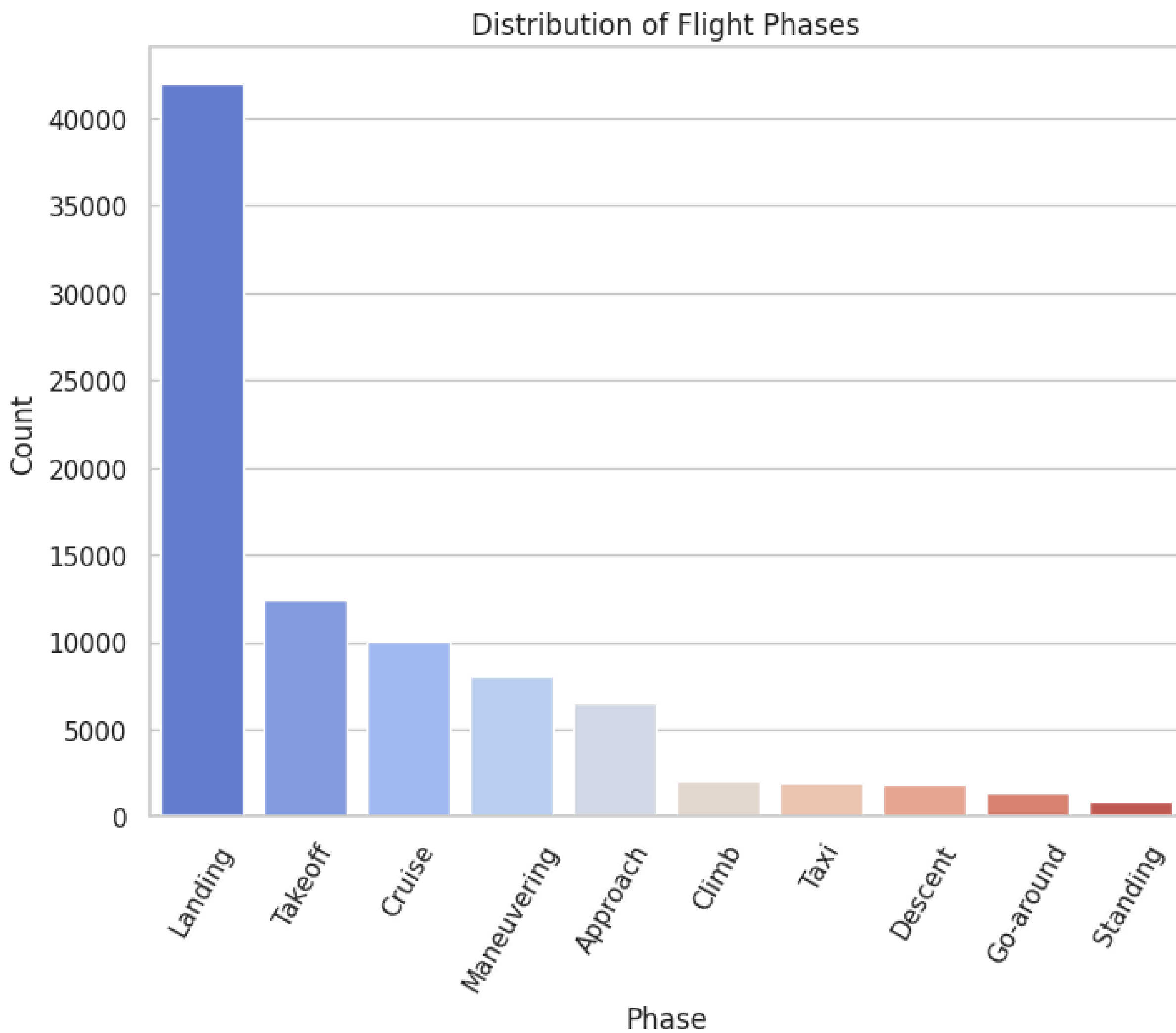
In [40]:
```
# Distribution of Number of Engines
num_engine_count = df['number_of_engines'].value_counts().head(10)
plt.figure(figsize=(8,6))
sns.barplot(x=num_engine_count.index,y=num_engine_count.values,palette='coolwarm')
plt.title('Distribution of Engines Number')
plt.xlabel('Number of Engines')
plt.ylabel('Count')
plt.xticks(rotation=60)
plt.show()
```



Observations

Most accidents involve aircraft with "1" engine, followed by "2".

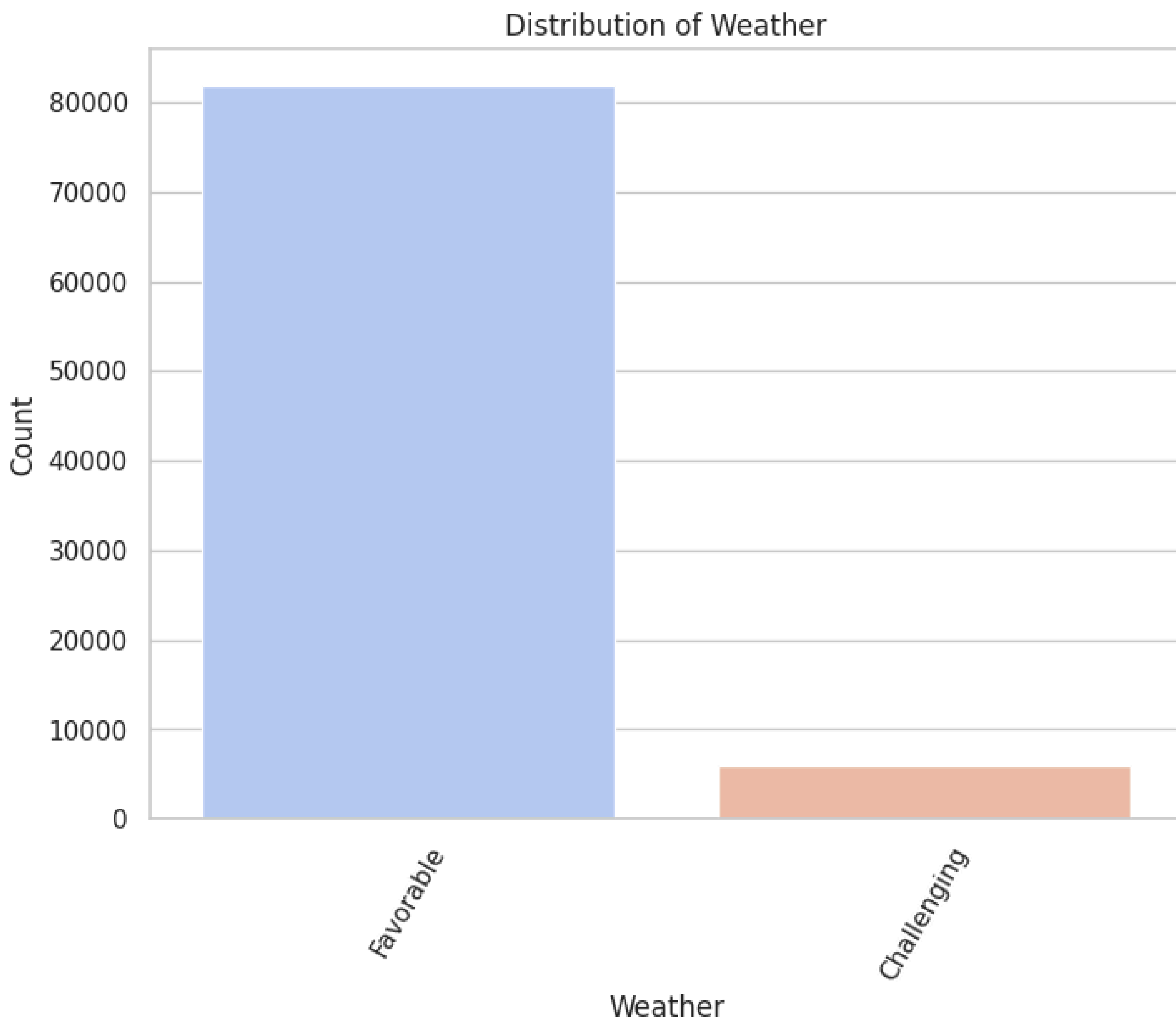
In [41]:
```
# Distribution of Broad Phase of Flight
flight_count = df['broad_phase_of_flight'].value_counts().head(10)
plt.figure(figsize=(8,6))
sns.barplot(x=flight_count.index,y=flight_count.values,palette='coolwarm')
plt.title('Distribution of Flight Phases')
plt.xlabel('Phase')
plt.ylabel('Count')
plt.xticks(rotation=60)
plt.show()
```



Observations

- The "Landing" phase has the highest number of accidents, followed closely by "Takeoff" and "Approach".
- Accidents are significantly less frequent during phases like "Climb", "Cruise", and "Taxi". This highlights that critical phases of flight (takeoff and landing) are associated with a higher risk of incidents.

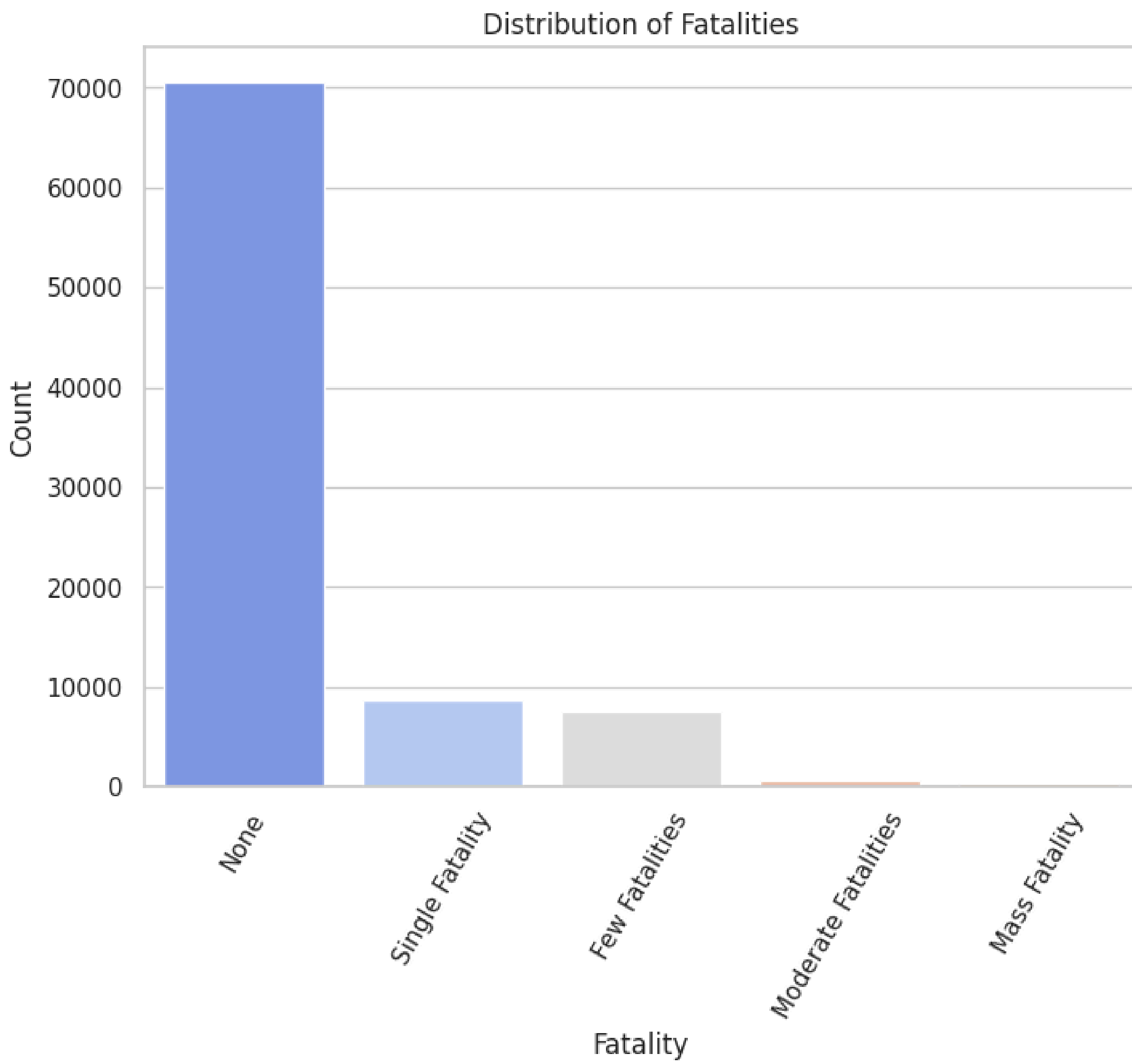
In [42]:
```
# Distribution of Weather
weather_count = df['weather_category'].value_counts()
plt.figure(figsize=(8,6))
sns.barplot(x=weather_count.index,y=weather_count.values,palette='coolwarm')
plt.title('Distribution of Weather')
plt.xlabel('Weather')
plt.ylabel('Count')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- Accidents occur more frequently under "VMC" (Favorable) weather conditions than "IMC" (Challenging) conditions.
- This might seem counterintuitive, but it could indicate that more flight operations generally occur under VMC, or that pilots might be more complacent or less prepared for issues in seemingly benign conditions, while being more cautious in IMC.

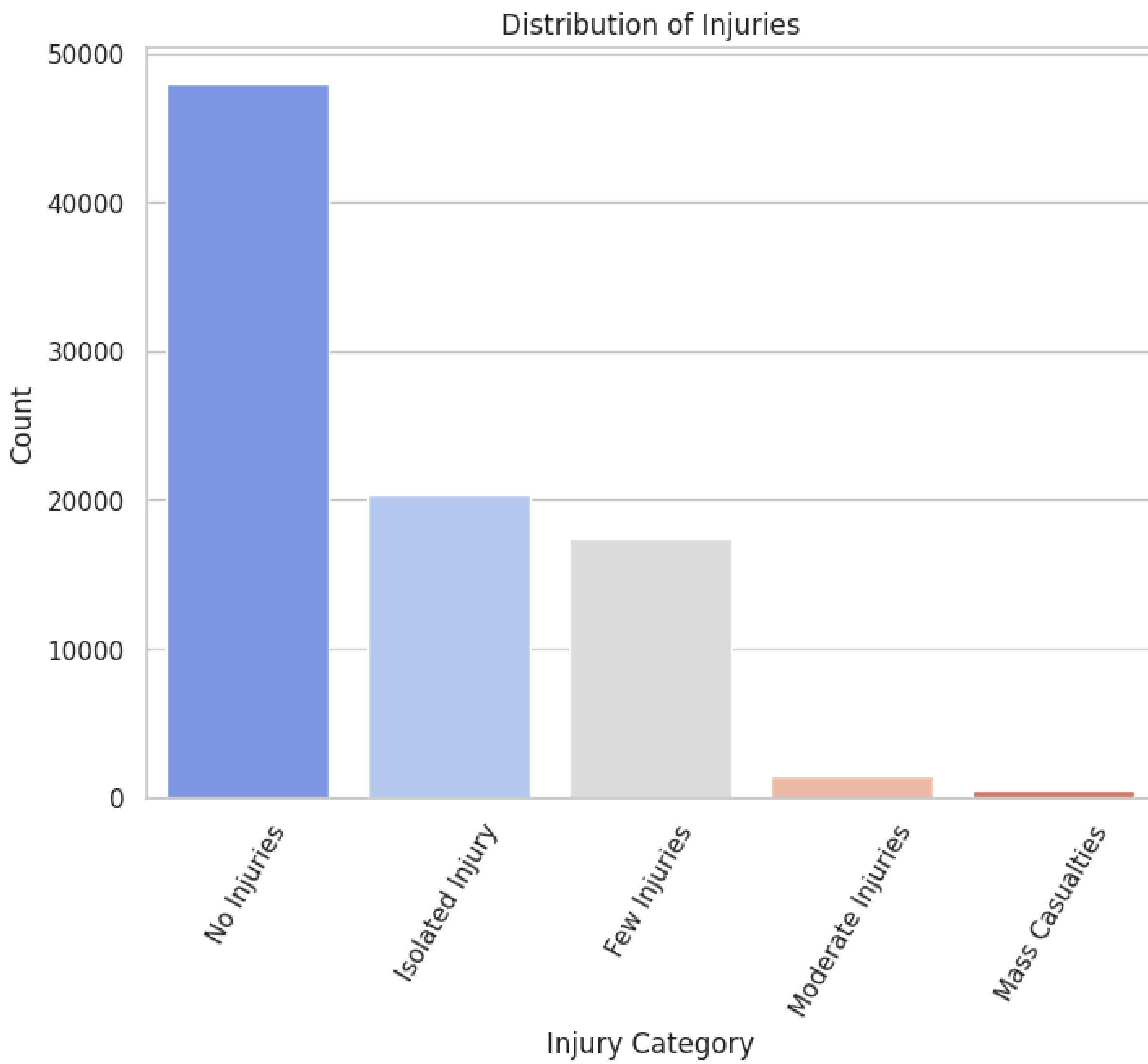
In [43]:
```
# Distribution of Fatalities
fatality_count = df['fatality_category'].value_counts()
plt.figure(figsize=(8,6))
sns.barplot(x=fatality_count.index,y=fatality_count.values,palette='coolwarm')
plt.title('Distribution of Fatalities')
plt.xlabel('Fatality')
plt.ylabel('Count')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- A large proportion of accidents result in "None" or "Single Fatality".
- "Mass Fatality" incidents are the least common, which aligns with overall aviation safety trends where accidents with high fatalities are rare.

In [44]:
```
# Distribution of Injuries
injury_count = df['injury_severity_category'].value_counts()
plt.figure(figsize=(8,6))
sns.barplot(x=injury_count.index,y=injury_count.values,palette='coolwarm')
plt.title('Distribution of Injuries')
plt.xlabel('Injury Category')
plt.ylabel('Count')
plt.xticks(rotation=60)
plt.show()
```



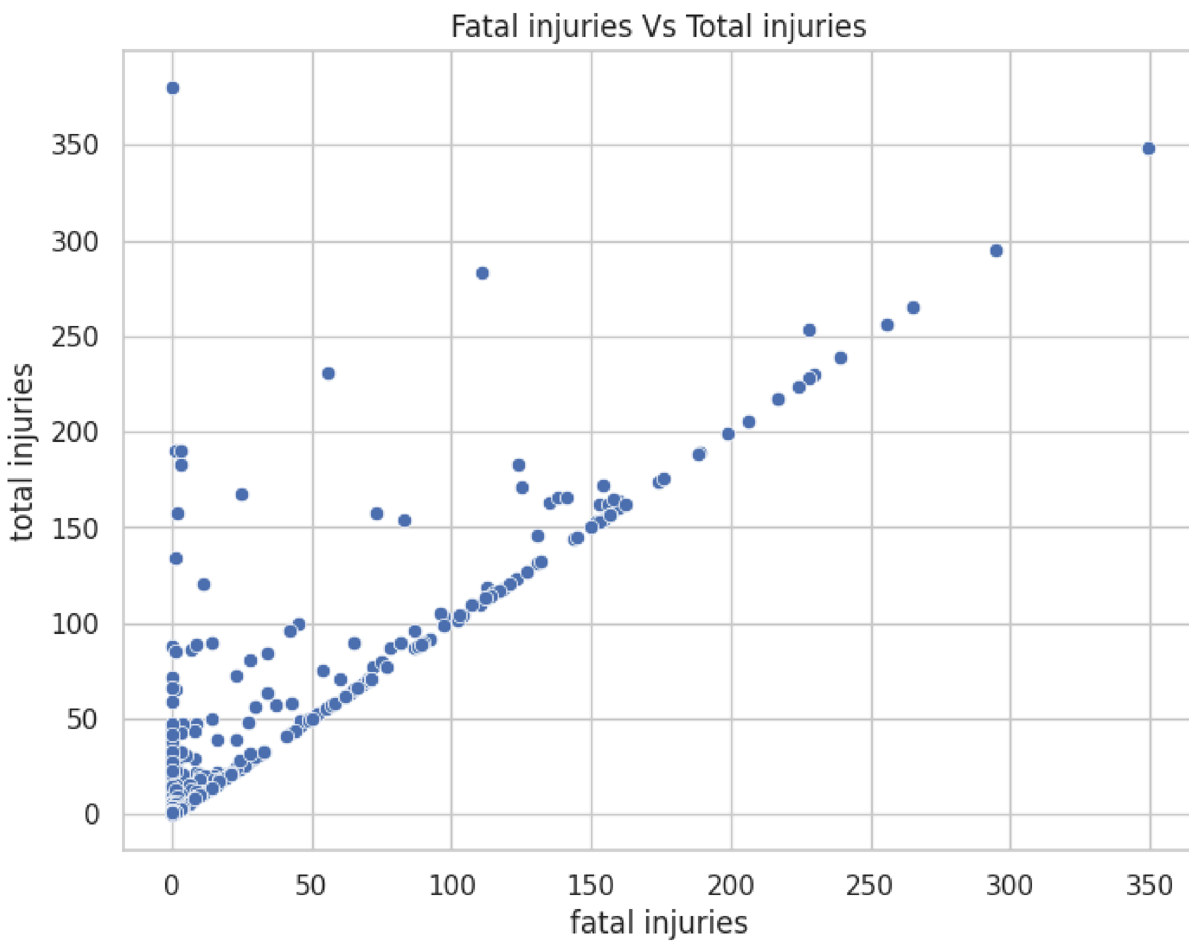
Observations:

- Similar to fatalities, the most frequent outcome in terms of overall injuries is "No Injuries", followed by "Isolated Injury".
- "Mass Casualties" are the least frequent outcome.

Bi-variate Analysis

Numeric Vs Numeric

In [45]:
```
# Fatal injuries vs Total injuries
plt.figure(figsize=(8,6))
sns.scatterplot(x='total_fatal_injuries',y='total_injuries',data=df)
plt.xlabel('fatal injuries')
plt.ylabel('total injuries')
plt.title('Fatal injuries Vs Total injuries')
plt.show()
```

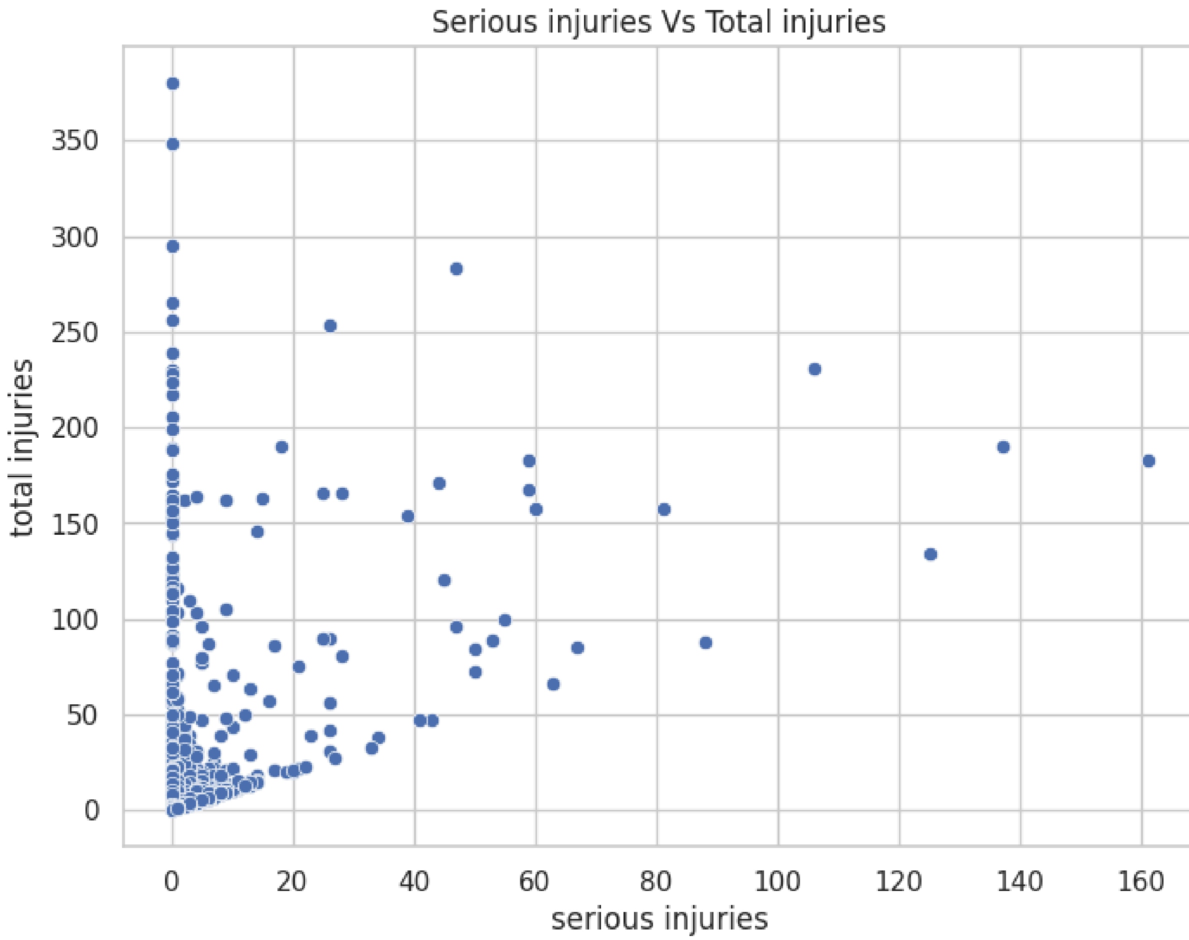


Observations:

- There is a strong positive correlation between total fatal injuries and total injuries. As the number of fatal injuries increases, the total number of injuries also tends to increase.
- Many points are clustered along the x-axis and close to the y-axis, indicating many accidents with few or no injuries.
- There's a clear line where total injuries equal fatal injuries, representing accidents where all injuries were fatal.

In [46]:

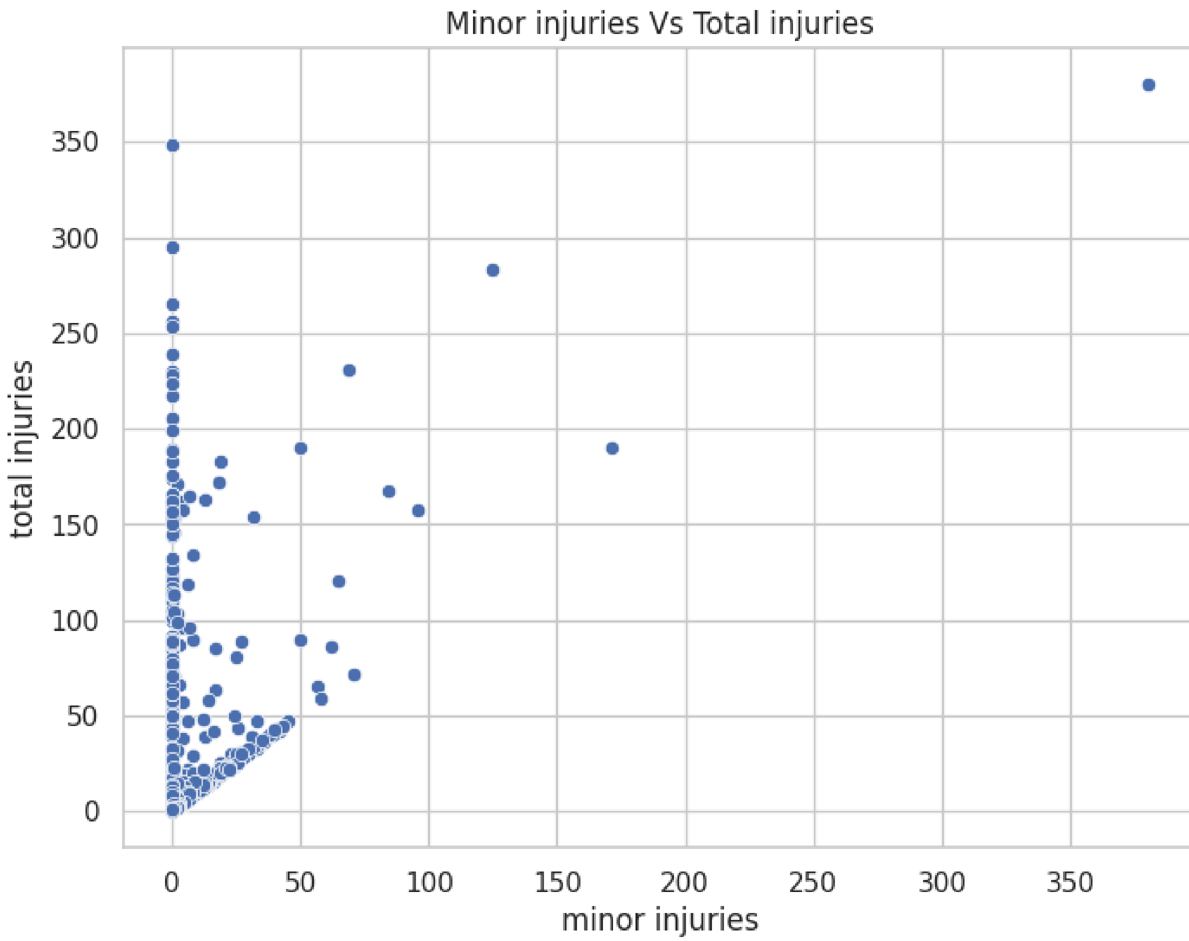
```
# Serious injuries vs Total injuries
plt.figure(figsize=(8,6))
sns.scatterplot(x='total_serious_injuries',y='total_injuries',data=df)
plt.xlabel('serious injuries')
plt.ylabel('total injuries')
plt.title('Serious injuries Vs Total injuries')
plt.show()
```



Observations:

- There is a positive correlation between serious injuries and total injuries, but it appears less strong than the correlation between fatal injuries and total injuries.
- Many data points show zero serious injuries across a range of total injuries.

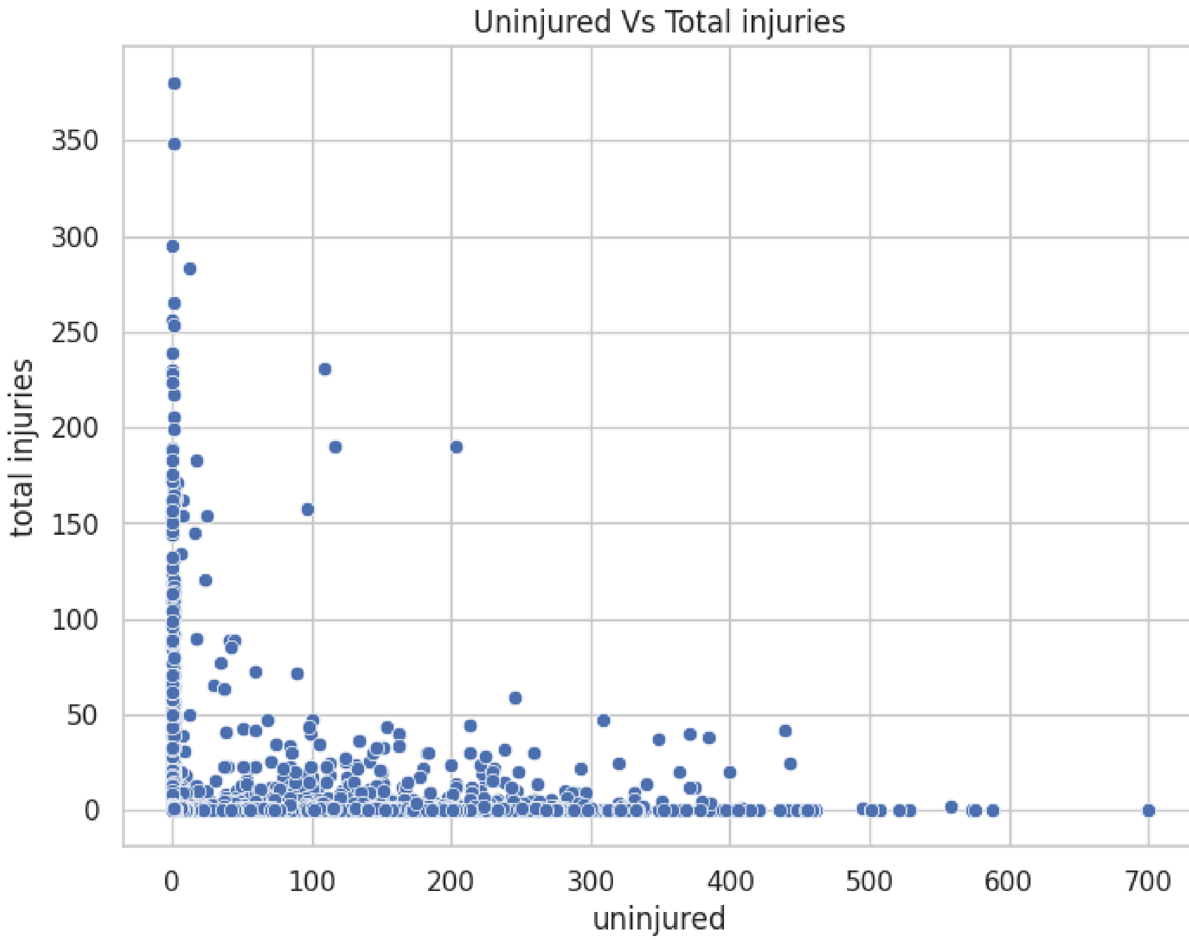
In [47]:
```
# Minor injuries vs Total injuries
plt.figure(figsize=(8,6))
sns.scatterplot(x='total_minor_injuries',y='total_injuries',data=df)
plt.xlabel('minor injuries')
plt.ylabel('total injuries')
plt.title('Minor injuries Vs Total injuries')
plt.show()
```



Observations:

- There is a positive relationship between minor injuries and total injuries.
- There are many accidents with zero minor injuries, especially when the total number of injuries is low.
- For accidents with higher total injuries, there's a greater likelihood of having minor injuries contributing to that total.

In [48]:
```
# Uninjured vs Total injuries
plt.figure(figsize=(8,6))
sns.scatterplot(x='total_uninjured',y='total_injuries',data=df)
plt.xlabel('uninjured')
plt.ylabel('total injuries')
plt.title('Uninjured Vs Total injuries')
plt.show()
```

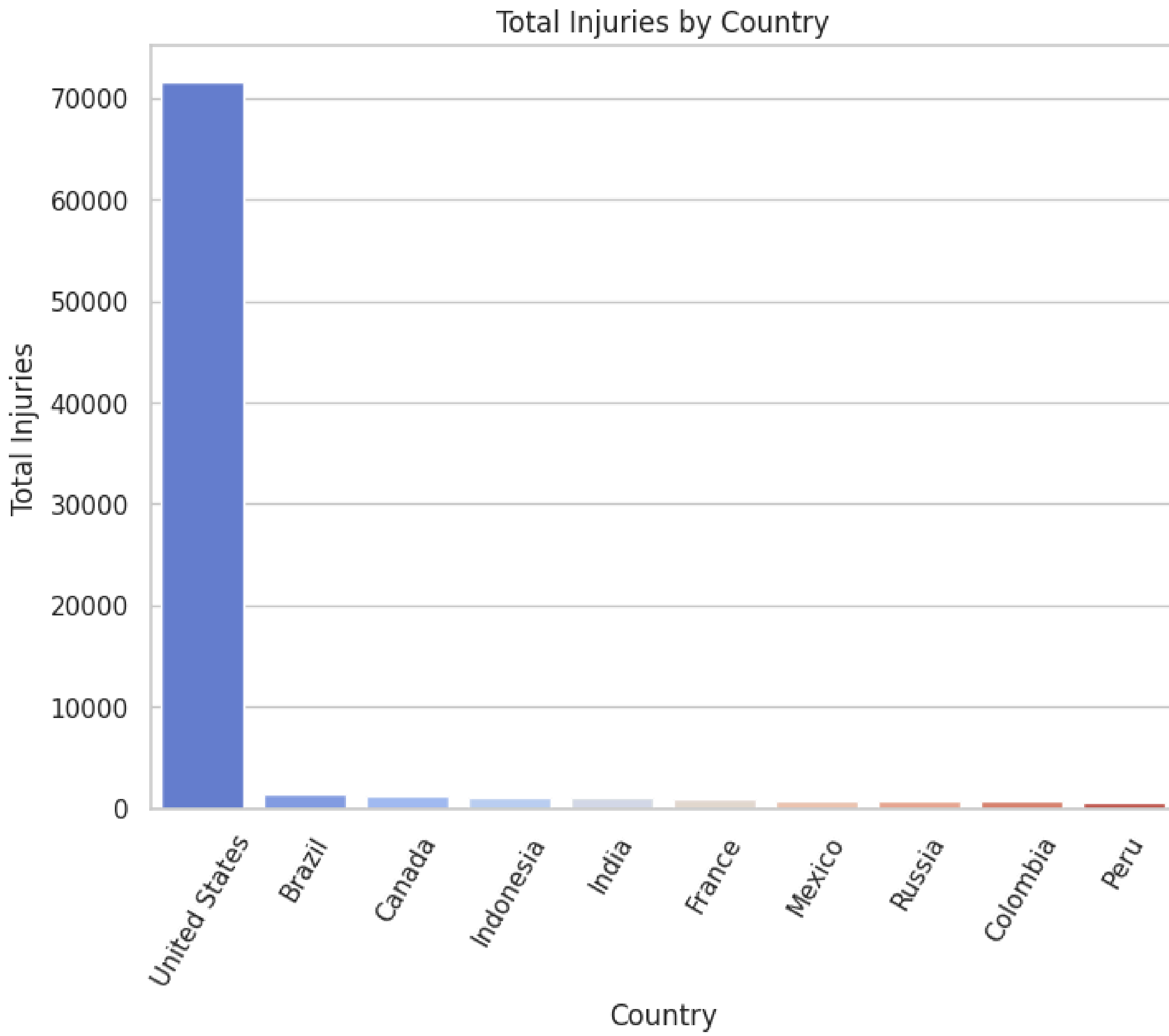


Observations:

- There isn't a clear positive or negative correlation between the number of uninjured individuals and the total number of injuries.
- Some accidents have a high number of uninjured people alongside a low number of total injuries, which might correspond to incidents where many people on board survived without injury.
- Conversely, accidents with a high number of total injuries tend to have a lower number of uninjured individuals.

Numeric Vs Categorical

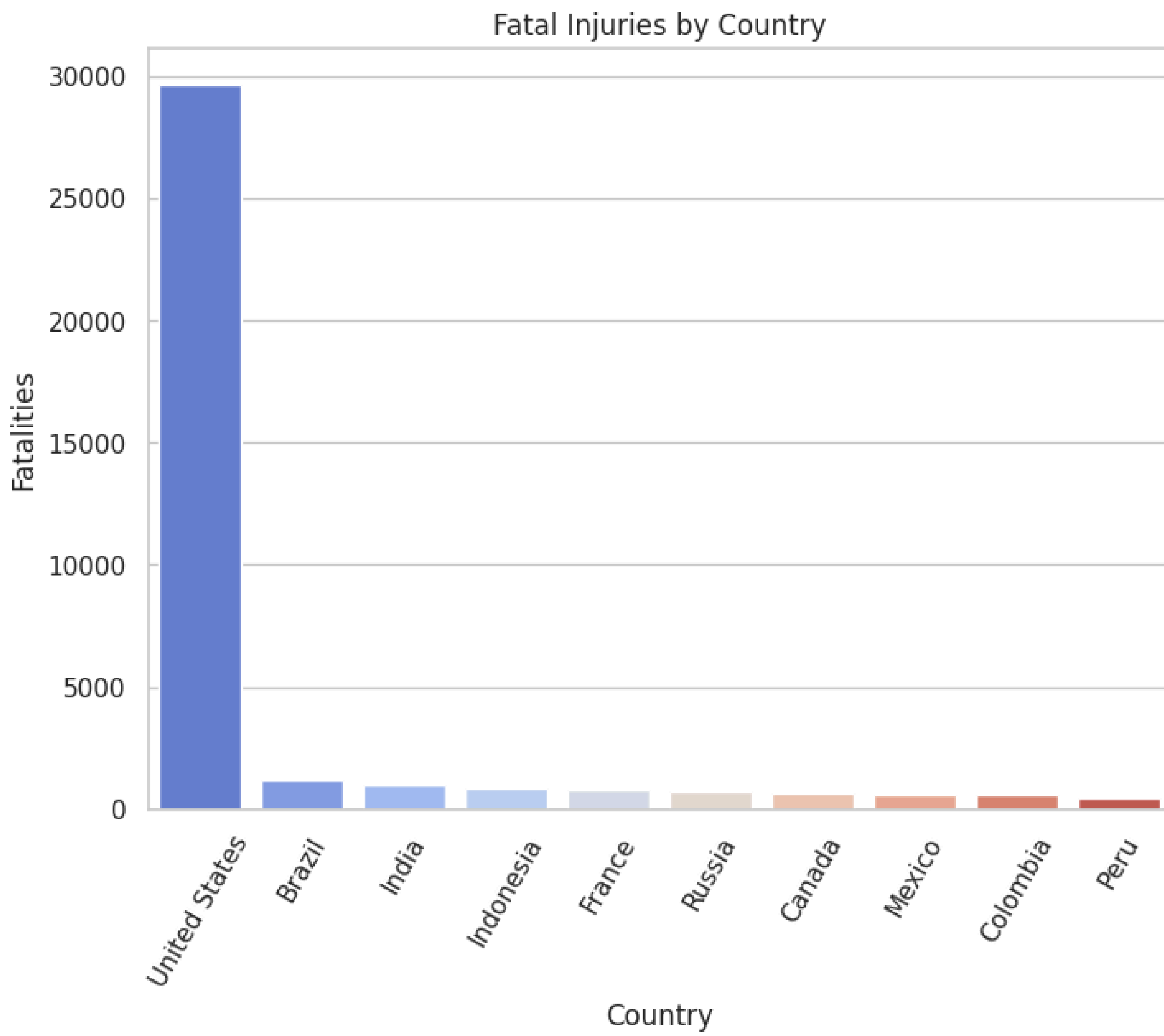
In [49]:
```
# Total Injuries Vs country
country_injuries = df.groupby('country')['total_injuries'].sum().sort_values(ascend
plt.figure(figsize=(8,6))
sns.barplot(x=country_injuries.index,y=country_injuries.values,palette='coolwarm')
plt.title('Total Injuries by Country')
plt.xlabel('Country')
plt.ylabel('Total Injuries')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- The United States accounts for the vast majority of total injuries from aviation accidents in the dataset. This is consistent with the fact that the dataset is primarily from the NTSB.
- Other countries have significantly fewer total injuries reported in this dataset.

In [50]:
```
# Fatal Injuries Vs country
fatal_country_injuries = df.groupby('country')['total_fatal_injuries'].sum().sort_v
plt.figure(figsize=(8,6))
sns.barplot(x=fatal_country_injuries.index,y=fatal_country_injuries.values,palette=
plt.title('Fatal Injuries by Country')
plt.xlabel('Country')
plt.ylabel('Fatalities')
plt.xticks(rotation=60)
plt.show()
```

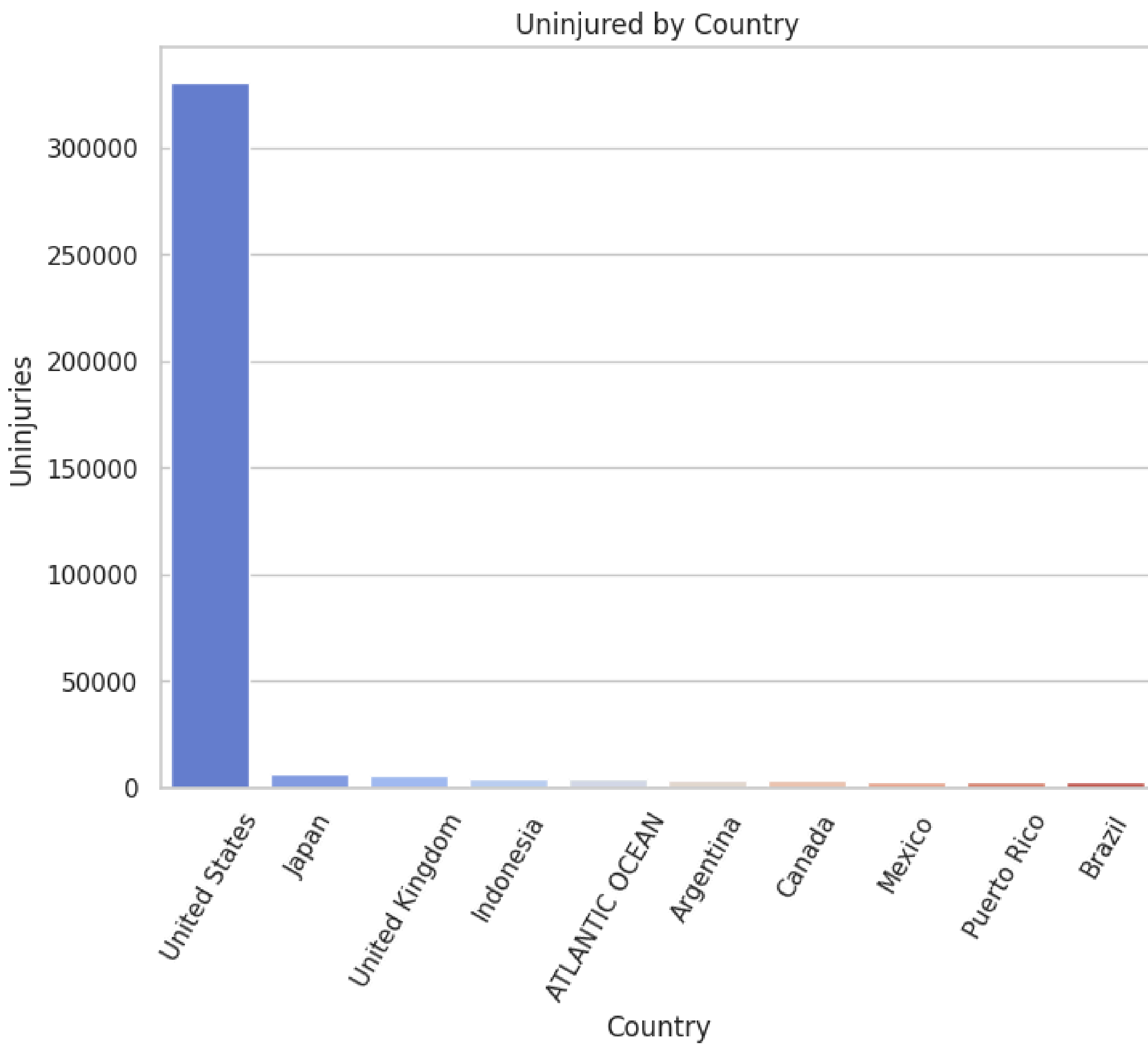


Observations:

- The United States also has the highest number of fatal injuries reported in the dataset, again due to the dataset's origin.
- The number of fatal injuries in other countries is much lower.

In [51]:

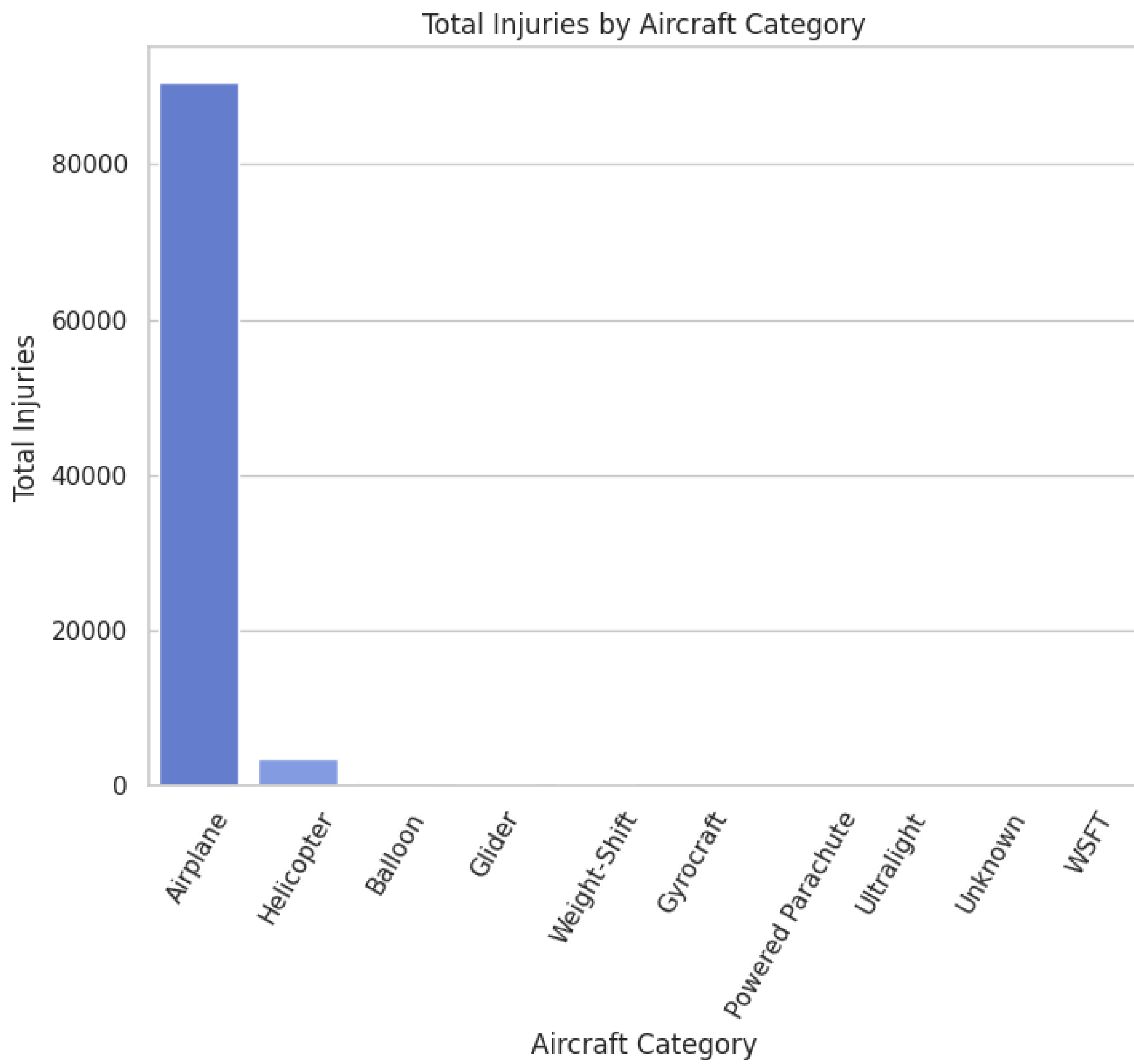
```
# Uninjured Vs country
country_uninjuried = df.groupby('country')['total_uninjured'].sum().sort_values(asc
plt.figure(figsize=(8,6))
sns.barplot(x=country_uninjuried.index,y=country_uninjuried.values,palette='coolwar
plt.title('Uninjured by Country')
plt.xlabel('Country')
plt.ylabel('Uninjuries')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- The United States has the highest number of uninjured individuals reported in accidents within this dataset. This aligns with the total number of accidents being highest in the US.

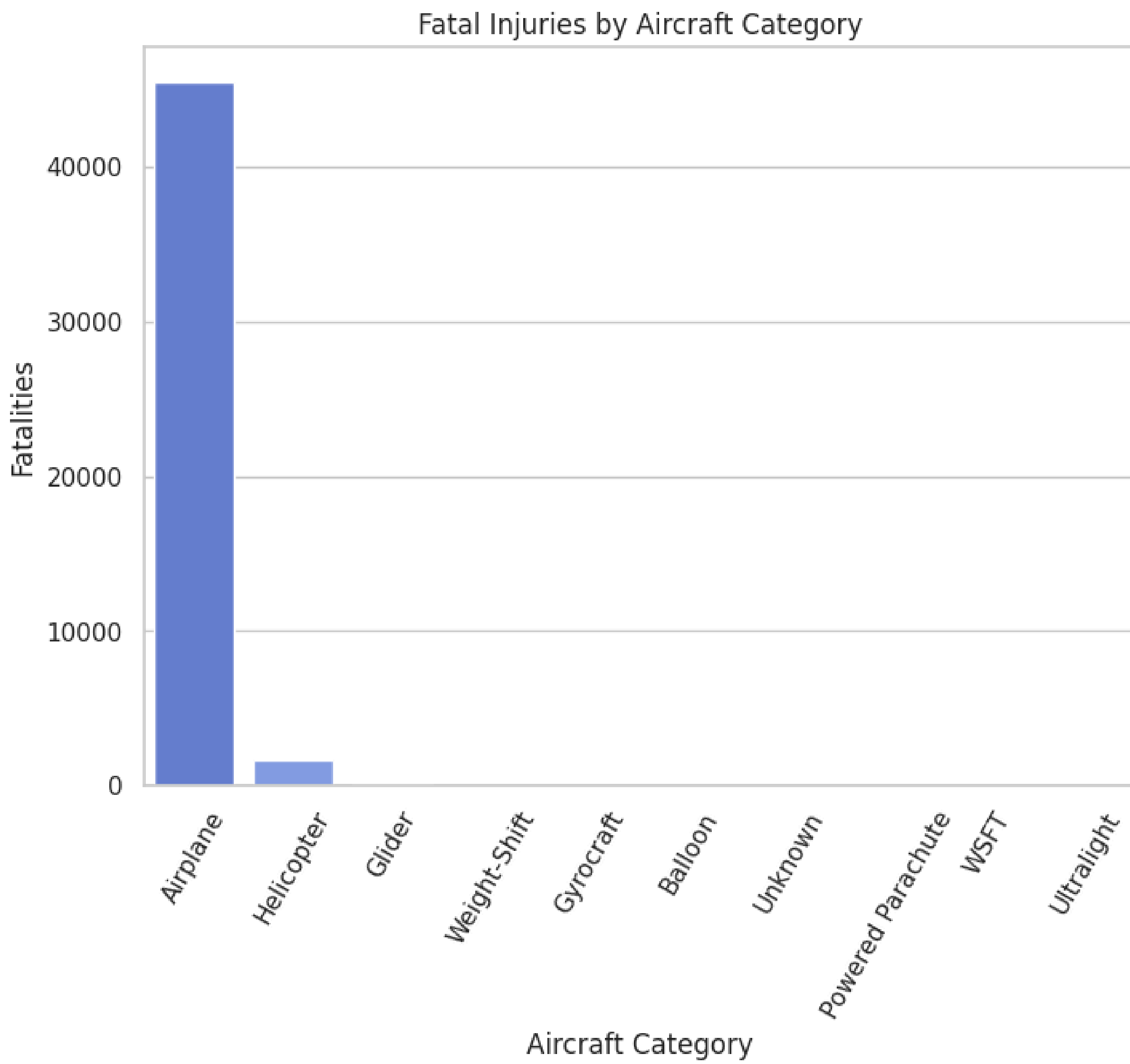
In [52]:
```
# Total Injuries Vs Aircraft Category
aircraft_injuries = df.groupby('aircraft_category')['total_injuries'].sum().sort_va
plt.figure(figsize=(8,6))
sns.barplot(x=aircraft_injuries.index,y=aircraft_injuries.values,palette='coolwarm'
plt.title('Total Injuries by Aircraft Category')
plt.xlabel('Aircraft Category')
plt.ylabel('Total Injuries')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- "Airplane" category accounts for the highest number of total injuries. This is likely because airplanes are the most common aircraft type and carry more passengers than smaller aircraft like helicopters or gliders.
- "Helicopter" and "Glider" categories have significantly fewer total injuries in comparison.

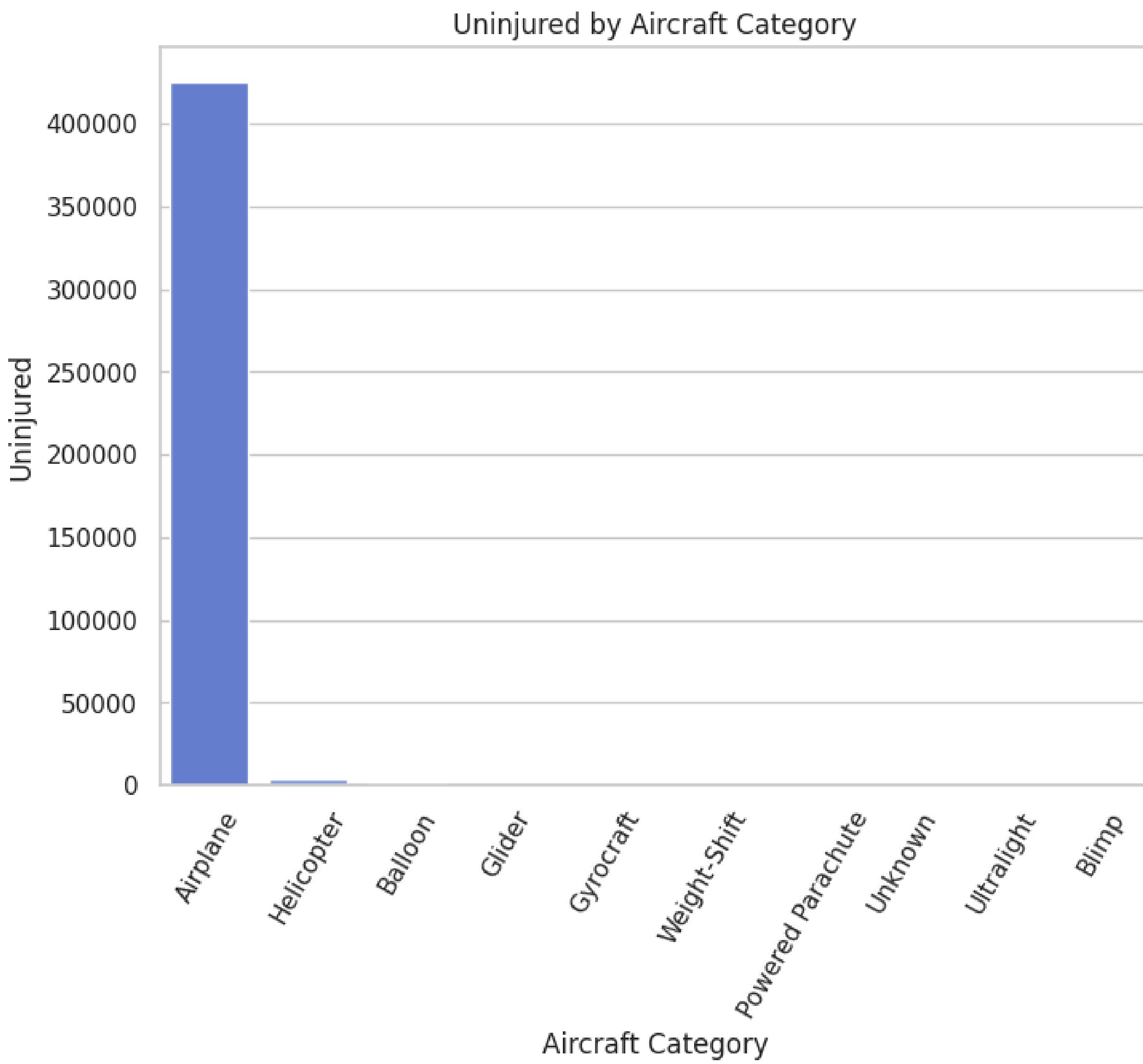
In [53]:
```
# fatal Injuries Vs Aircraft Category
aircraft_fatal_injuries = df.groupby('aircraft_category')['total_fatal_injuries'].s
plt.figure(figsize=(8,6))
sns.barplot(x=aircraft_fatal_injuries.index,y=aircraft_fatal_injuries.values,palett
plt.title('Fatal Injuries by Aircraft Category')
plt.xlabel('Aircraft Category')
plt.ylabel('Fatalities')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- "Airplane" category also accounts for the highest number of fatal injuries.
- The number of fatal injuries in "Helicopter" accidents is considerably lower than in "Airplane" accidents but still notable.

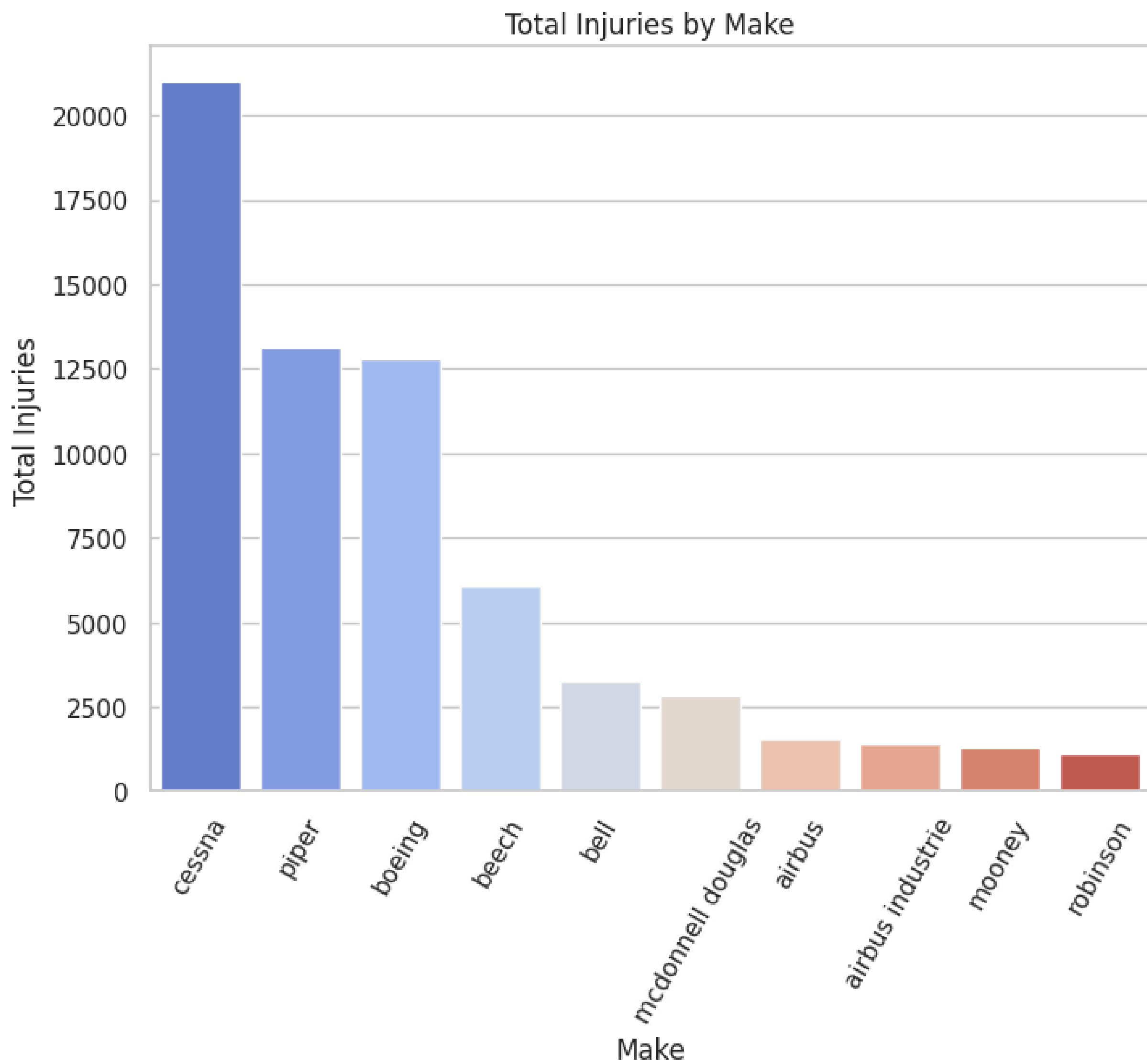
In [54]:
```
# Uninjured Vs Aircraft Category
aircraft_uninjuried = df.groupby('aircraft_category')['total_uninjured'].sum().sort
plt.figure(figsize=(8,6))
sns.barplot(x=aircraft_uninjuried.index,y=aircraft_uninjuried.values,palette='coolw
plt.title('Uninjured by Aircraft Category')
plt.xlabel('Aircraft Category')
plt.ylabel('Uninjured')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- "Airplane" category has the highest number of uninjured individuals, which is expected given they have higher passenger capacities.

In [55]:
```
# Total Injuries Vs Make
make_injuries = df.groupby('make')['total_injuries'].sum().sort_values(ascending=Fa
plt.figure(figsize=(8,6))
sns.barplot(x=make_injuries.index,y=make_injuries.values,palette='coolwarm')
plt.title('Total Injuries by Make')
plt.xlabel('Make')
plt.ylabel('Total Injuries')
plt.xticks(rotation=60)
plt.show()
```



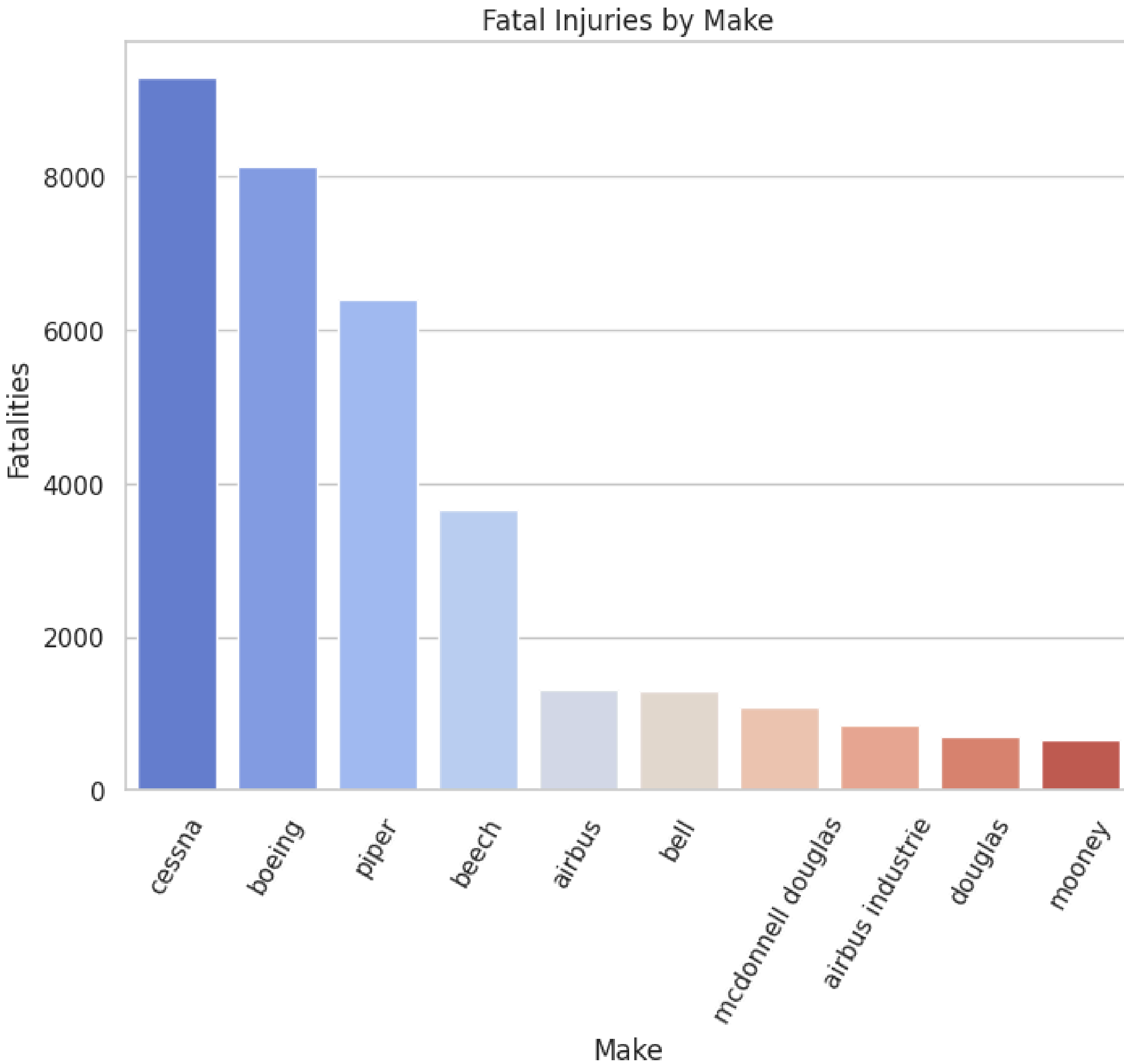
Observations:

- Cessna has the most injuries by far.
- Piper and Boeing are the next two highest, closely ranked.
- There's a sharp drop in injuries after the top three manufacturers.

Mooney and Robinson have the fewest injuries among those shown.

In [56]:

```
#Total Fatal Injuries by Make
make_fatal_injuries = df.groupby('make')['total_fatal_injuries'].sum().sort_values(
plt.figure(figsize=(8,6))
sns.barplot(x=make_fatal_injuries.index,y=make_fatal_injuries.values,palette='coolw
plt.title('Fatal Injuries by Make')
plt.xlabel('Make')
plt.ylabel('Fatalities')
plt.xticks(rotation=60)
plt.show()
```

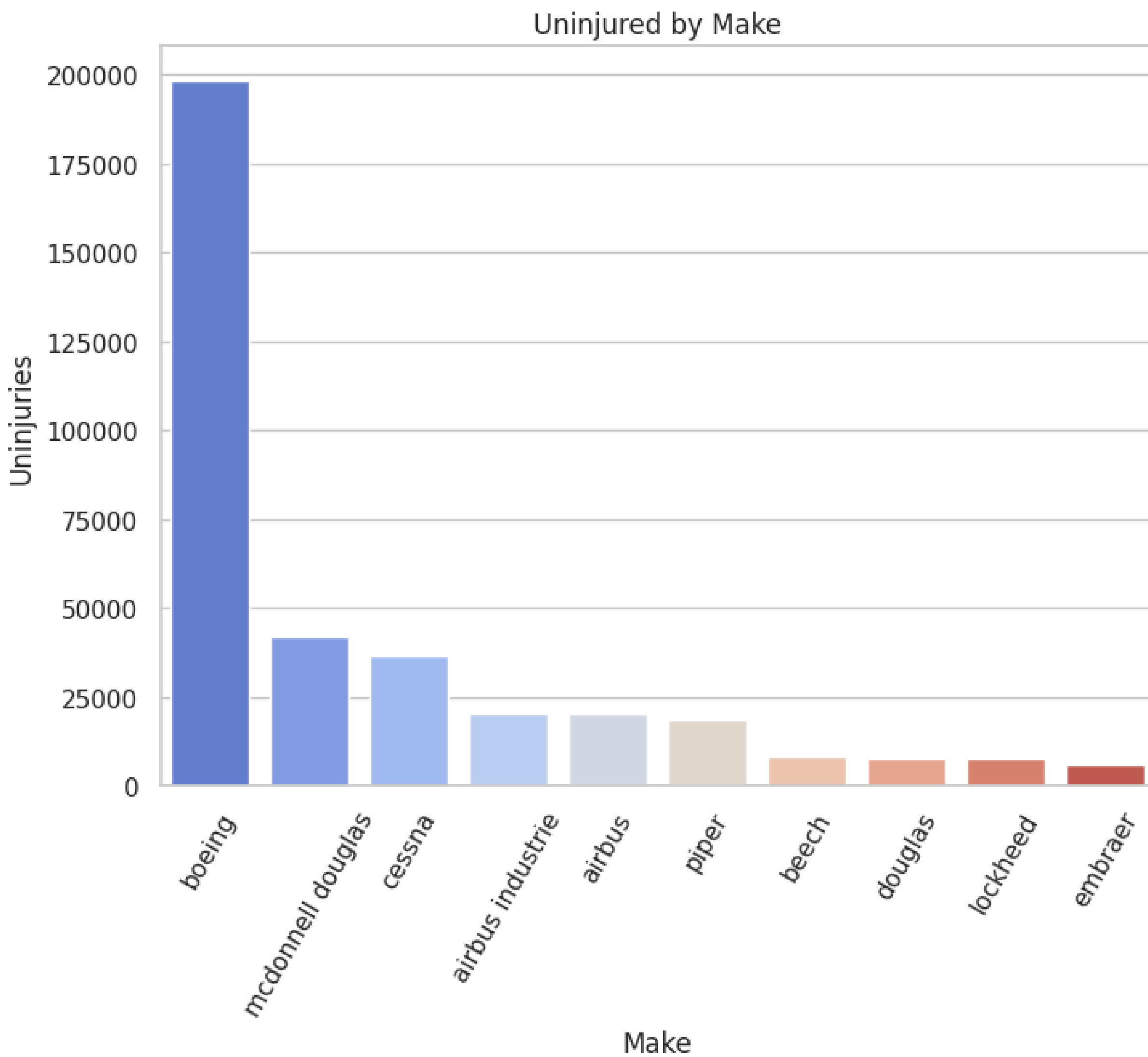


Observations:

- Cessna has the most fatalities, nearing 9,000.
- Boeing is second-highest, with Piper ranking third.
- There's a significant drop in fatalities after the top three (Cessna, Boeing, Piper).
- Mooney has the fewest fatalities among the listed makes.

In [57]:

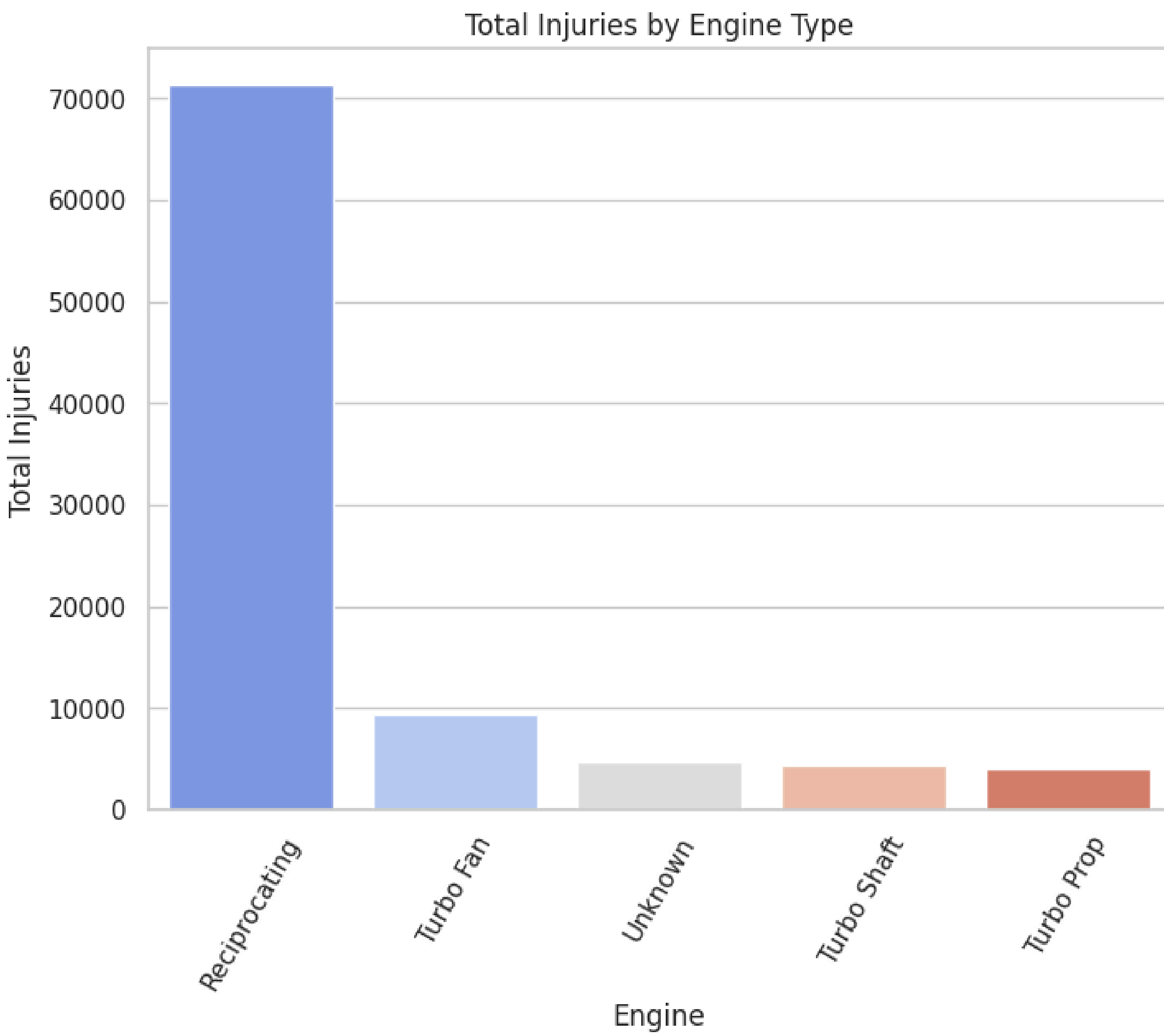
```
#Uninjured by Make
make_uninjured = df.groupby('make')['total_uninjured'].sum().sort_values(ascending=
plt.figure(figsize=(8,6))
sns.barplot(x=make_uninjured.index,y=make_uninjured.values,palette='coolwarm')
plt.title('Uninjured by Make')
plt.xlabel('Make')
plt.ylabel('Uninjuries')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- "Boeing" aircraft have the highest number of uninjured individuals, which is consistent with them having larger passenger capacities in the accidents recorded.

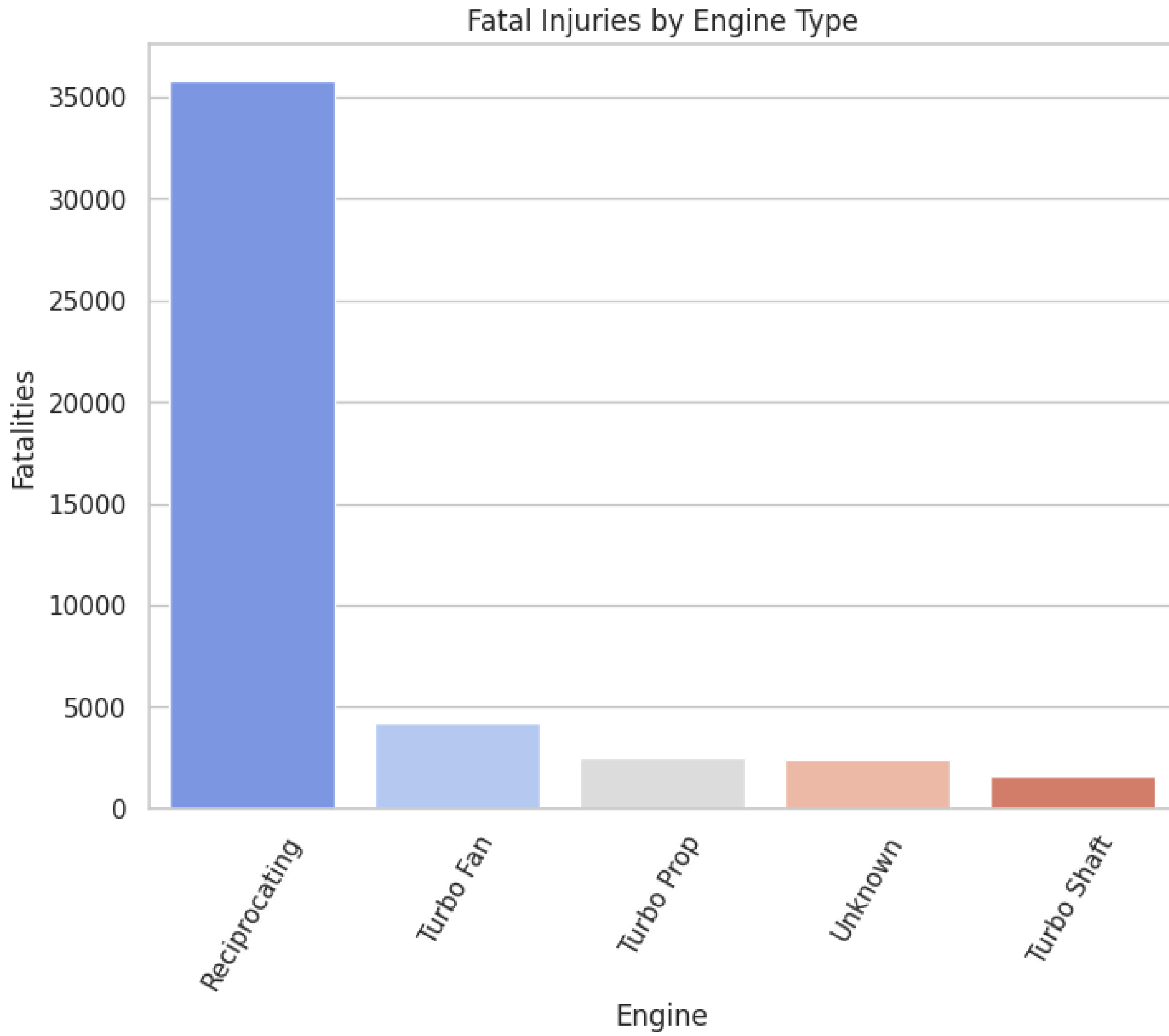
In [58]:
```
# Total Injuries Vs Engine Type
engine_injuries = df.groupby('engine_type')['total_injuries'].sum().sort_values(asc
plt.figure(figsize=(8,6))
sns.barplot(x=engine_injuries.index,y=engine_injuries.values,palette='coolwarm')
plt.title('Total Injuries by Engine Type')
plt.xlabel('Engine')
plt.ylabel('Total Injuries')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- Aircraft with "Jet" engines account for the highest number of total injuries. This is because jet engines power larger commercial aircraft.
- "Reciprocating" engine aircraft, while frequent in overall accidents, have fewer total injuries per incident as they are generally found in smaller aircraft.

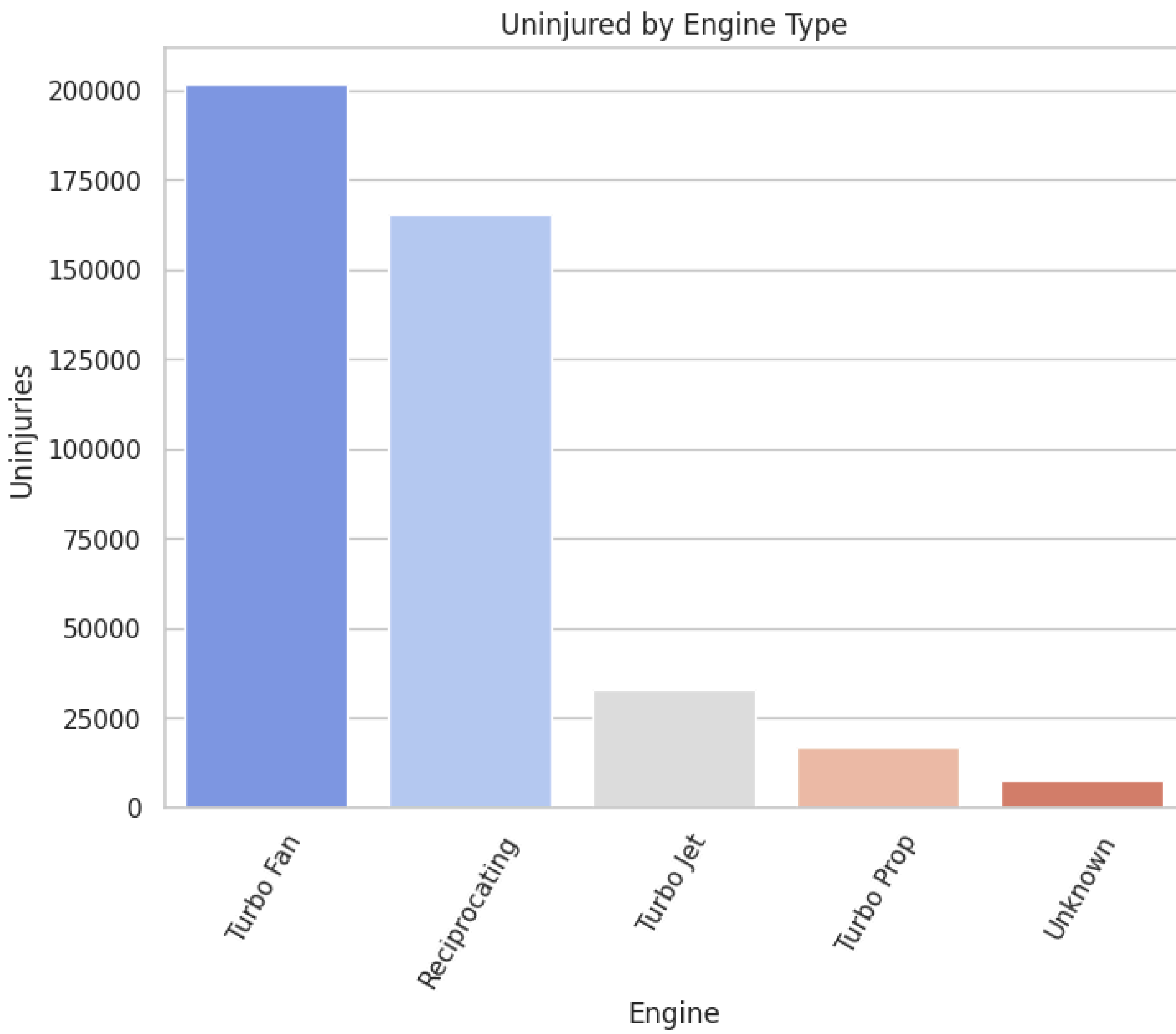
In [59]:
```
# Fatal Injuries Vs Engine Type
engine_fatal_injuries = df.groupby('engine_type')['total_fatal_injuries'].sum().sor
plt.figure(figsize=(8,6))
sns.barplot(x=engine_fatal_injuries.index,y=engine_fatal_injuries.values,palette='c
plt.title('Fatal Injuries by Engine Type')
plt.xlabel('Engine')
plt.ylabel('Fatalities')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- Aircraft with "Jet" engines are associated with the highest number of fatal injuries, aligning with the total injury trend for this category.
- "Reciprocating" engines are also involved in a significant number of fatal injuries, reflecting the high overall accident count for this engine type.

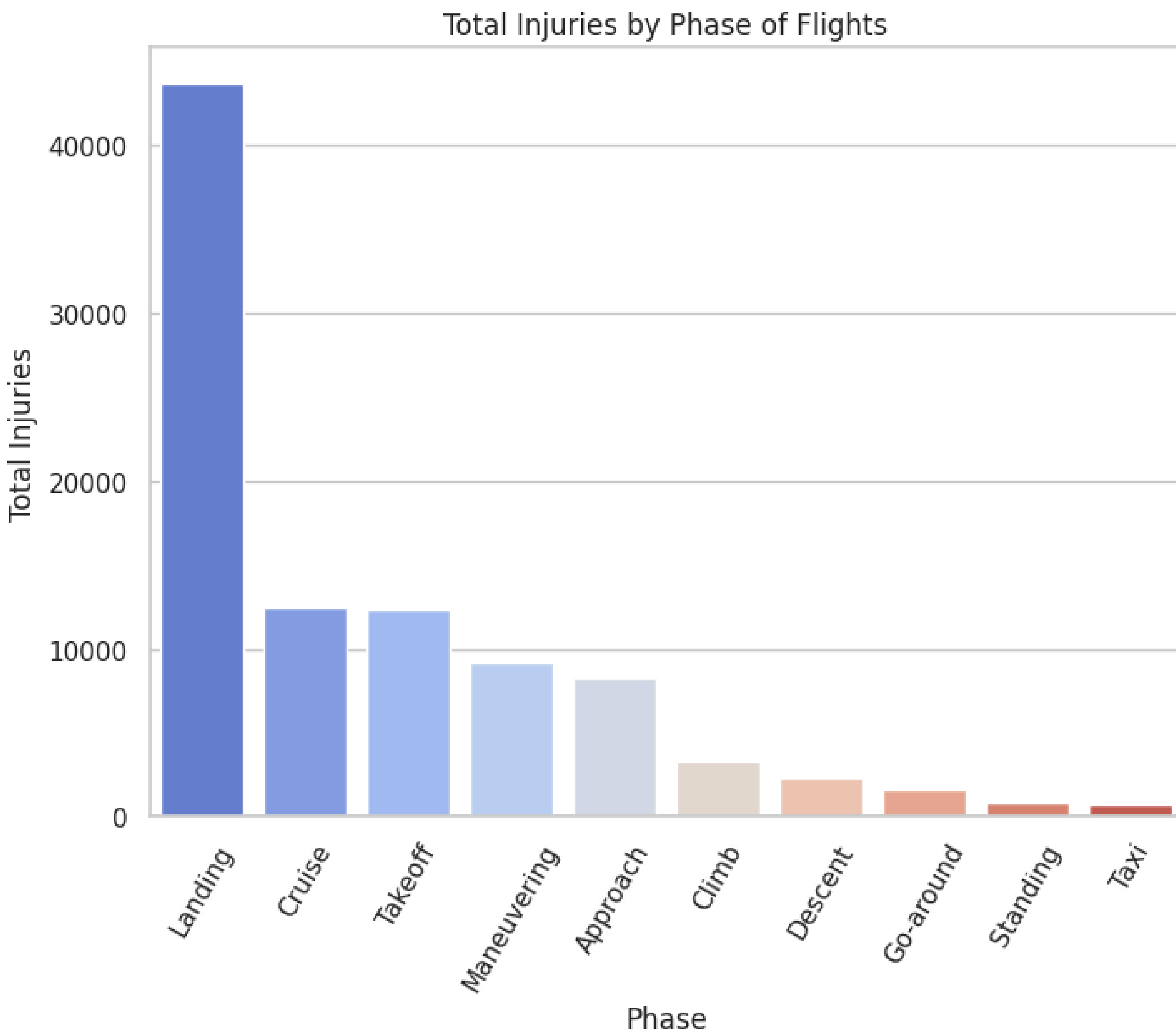
In [60]:
```
# Uninjured Vs Engine Type
engine_uninjured = df.groupby('engine_type')['total_uninjured'].sum().sort_values(a
plt.figure(figsize=(8,6))
sns.barplot(x=engine_uninjured.index,y=engine_uninjured.values,palette='coolwarm')
plt.title('Uninjured by Engine Type')
plt.xlabel('Engine')
plt.ylabel('Uninjuries')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- Aircraft with "Jet" engines have the highest number of uninjured individuals, due to the larger capacity of these aircraft.

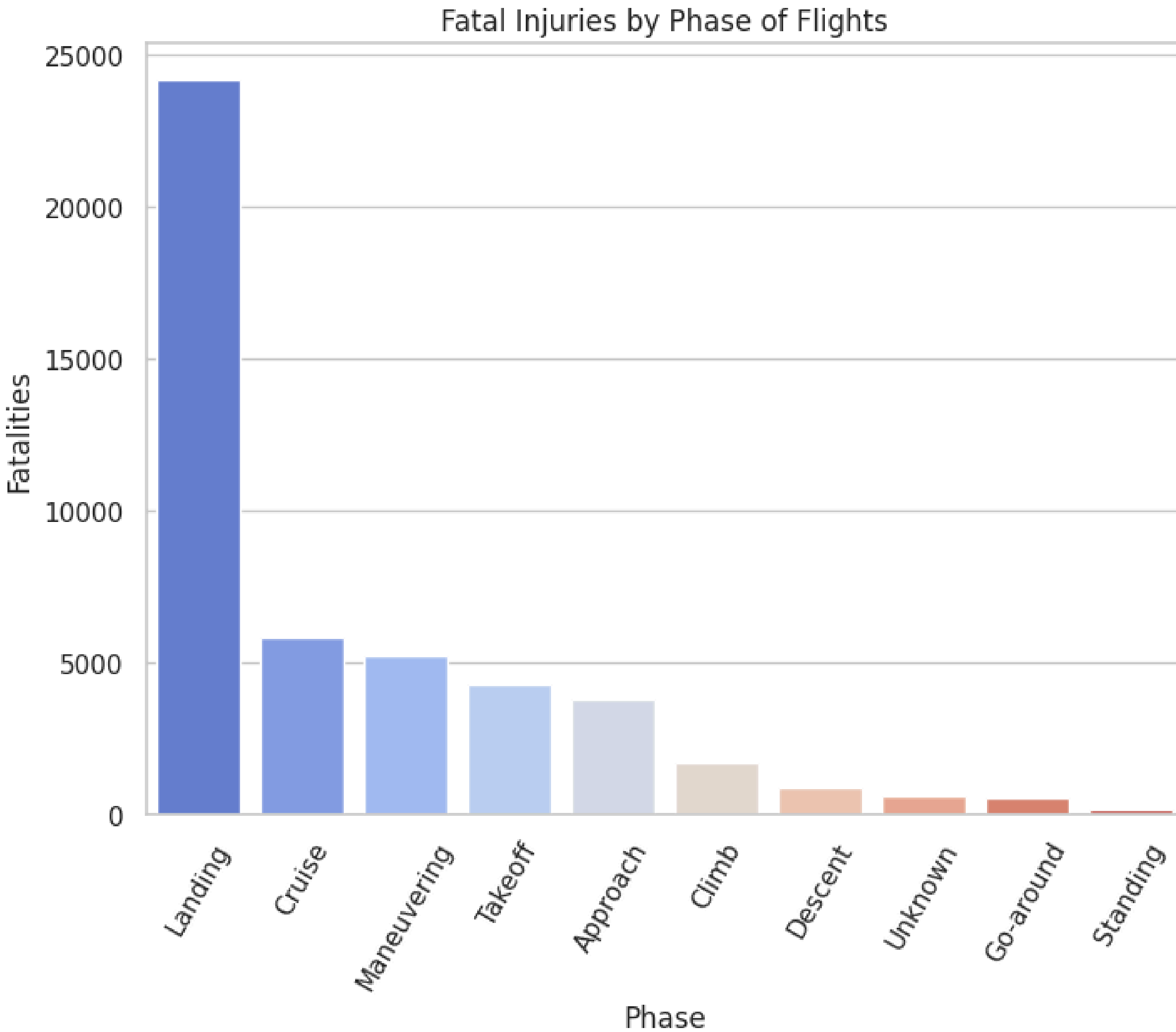
In [61]:
```
# Total Injuries Vs Phase of Flight
phase_injuries = df.groupby('broad_phase_of_flight')['total_injuries'].sum().sort_v
plt.figure(figsize=(8,6))
sns.barplot(x=phase_injuries.index,y=phase_injuries.values,palette='coolwarm')
plt.title('Total Injuries by Phase of Flights')
plt.xlabel('Phase')
plt.ylabel('Total Injuries')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- The "Landing" phase has the highest number of total injuries, closely followed by "Takeoff" and "Approach".
- While the "Cruise" phase has fewer total accidents, accidents during this phase can still result in a notable number of total injuries.

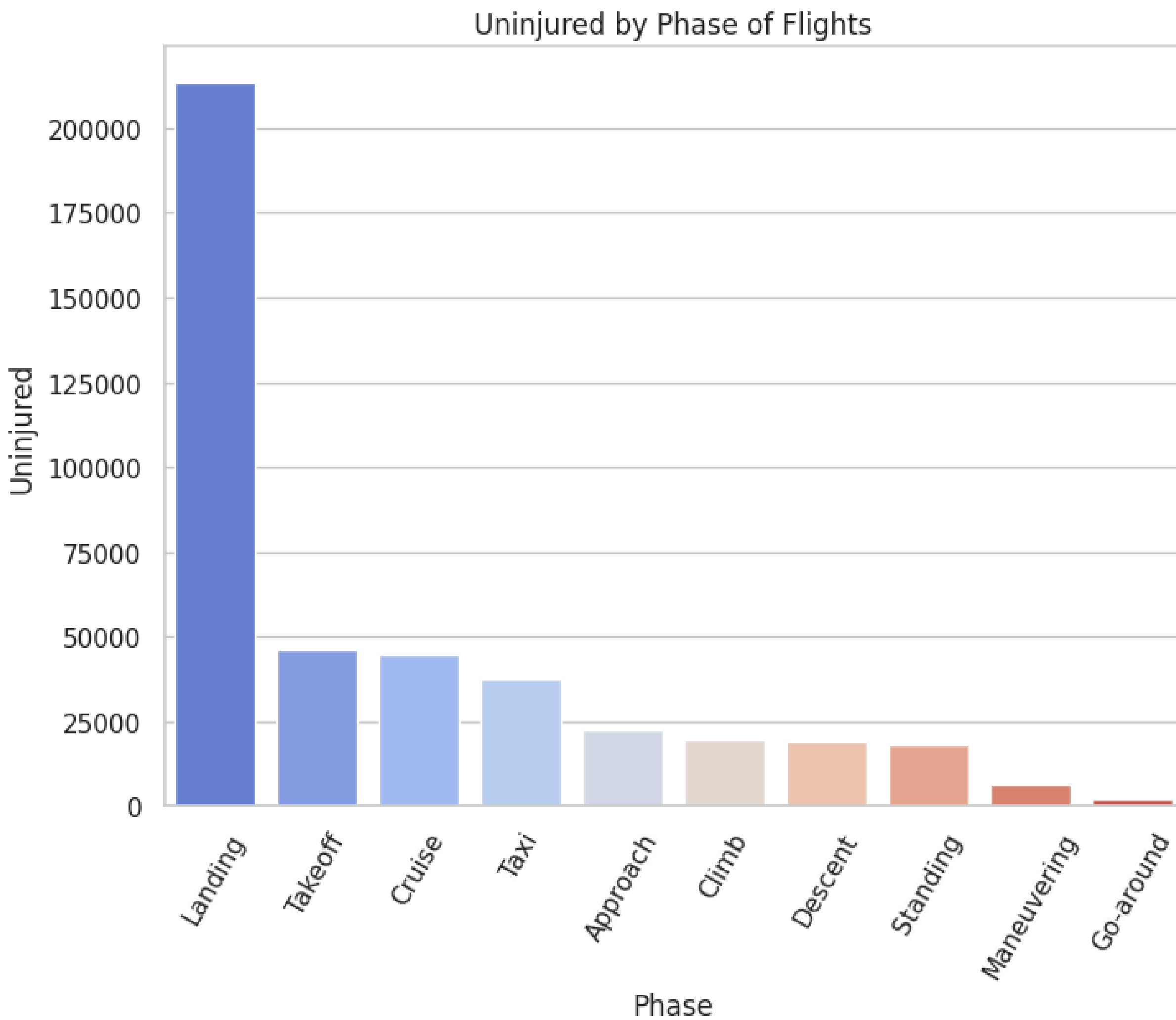
In [62]:
```
# Total Injuries Vs Phase of Flight
phase_fatal_injuries = df.groupby('broad_phase_of_flight')['total_fatal_injuries'].
plt.figure(figsize=(8,6))
sns.barplot(x=phase_fatal_injuries.index,y=phase_fatal_injuries.values,palette='coo
plt.title('Fatal Injuries by Phase of Flights')
plt.xlabel('Phase')
plt.ylabel('Fatalities')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- The "Landing" phase is associated with the highest number of fatal injuries, closely followed by "Takeoff" and "Approach".
- The "Cruise" phase also contributes a significant number of fatal injuries, despite having fewer accidents overall compared to the takeoff and landing phases.

In [63]:
```
# Uninjured Vs Phase of Flight
phase_uninjured = df.groupby('broad_phase_of_flight')['total_uninjured'].sum().sort
plt.figure(figsize=(8,6))
sns.barplot(x=phase_uninjured.index,y=phase_uninjured.values,palette='coolwarm')
plt.title('Uninjured by Phase of Flights')
plt.xlabel('Phase')
plt.ylabel('Uninjured')
plt.xticks(rotation=60)
plt.show()
```

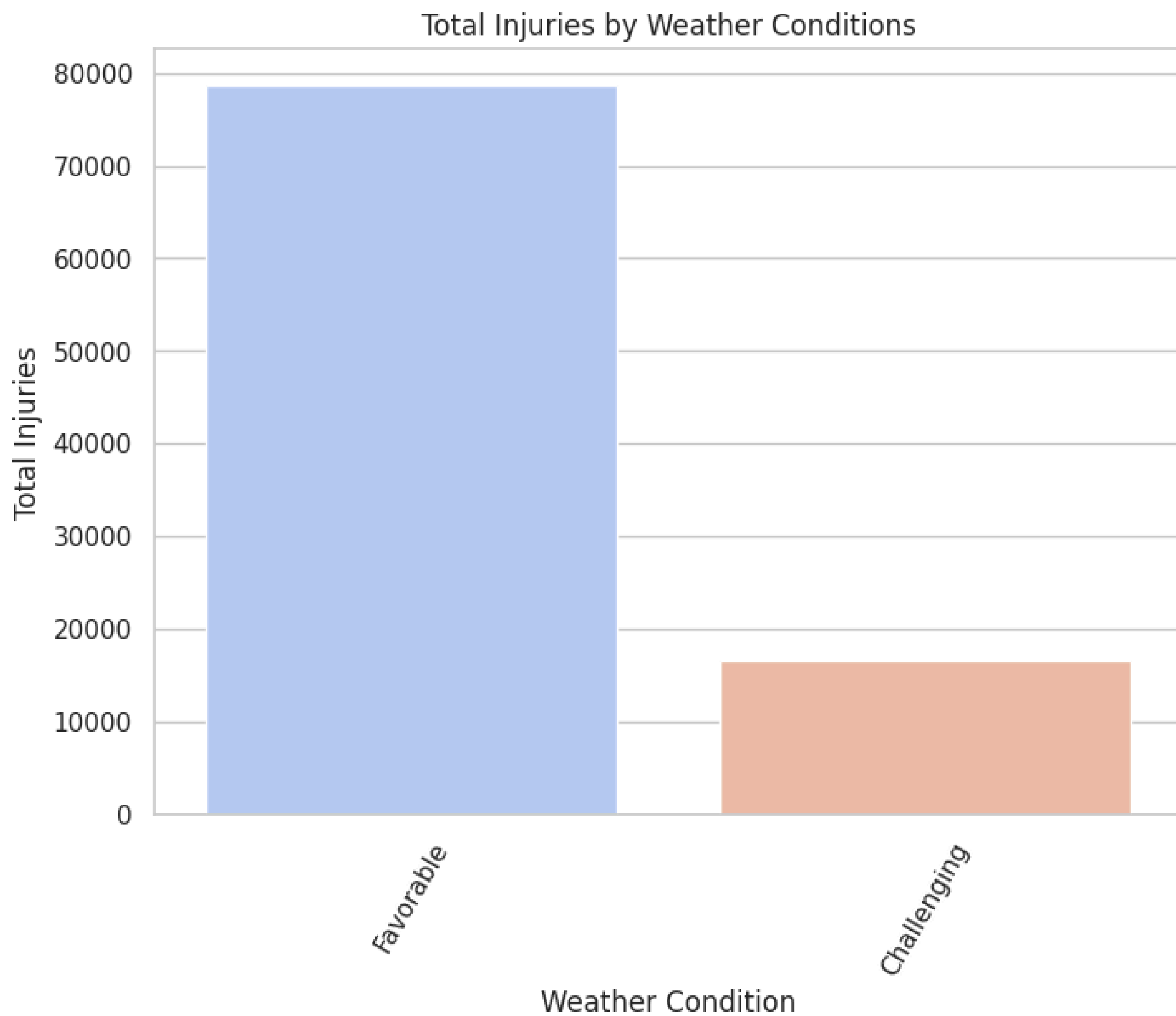


Observations:

- The "Landing", "Takeoff", and "Approach" phases show the highest numbers of uninjured individuals, consistent with these phases having the highest accident counts where occupants might survive without injury.
- The "Cruise" phase also has a notable number of uninjured individuals, likely from accidents involving larger aircraft with more occupants.

In [64]:

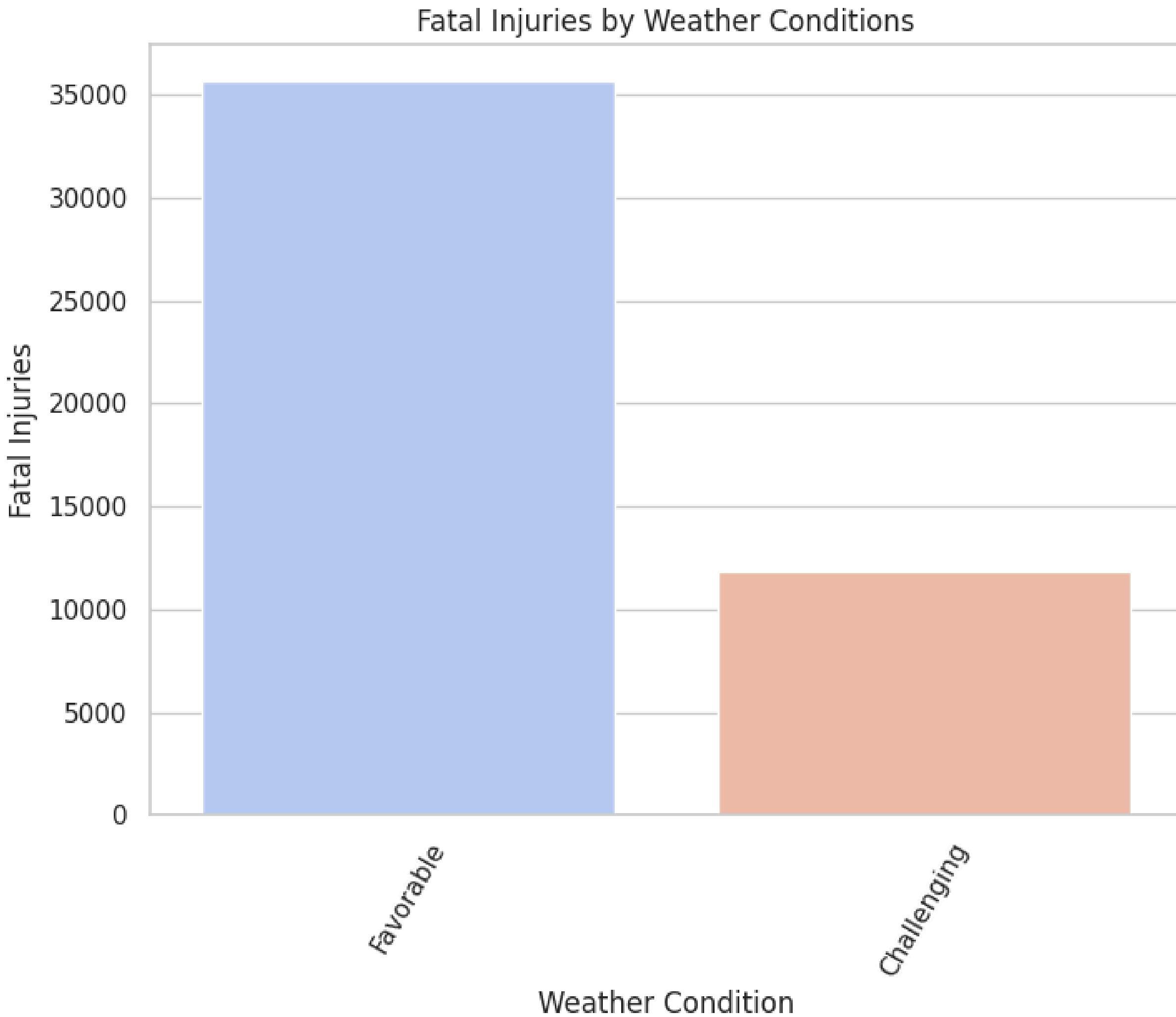
```
# Total Injuries Vs Weather Conditions
weather_injuries = df.groupby('weather_category')['total_injuries'].sum().sort_valu
plt.figure(figsize=(8,6))
sns.barplot(x=weather_injuries.index,y=weather_injuries.values,palette='coolwarm')
plt.title('Total Injuries by Weather Conditions')
plt.xlabel('Weather Condition')
plt.ylabel('Total Injuries')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- "VMC" (Favorable) weather conditions are associated with a significantly higher number of total injuries than
- "IMC" (Challenging) conditions. This is consistent with the observation that more accidents occur in VMC, and potentially indicates severe accidents can still happen in good weather.

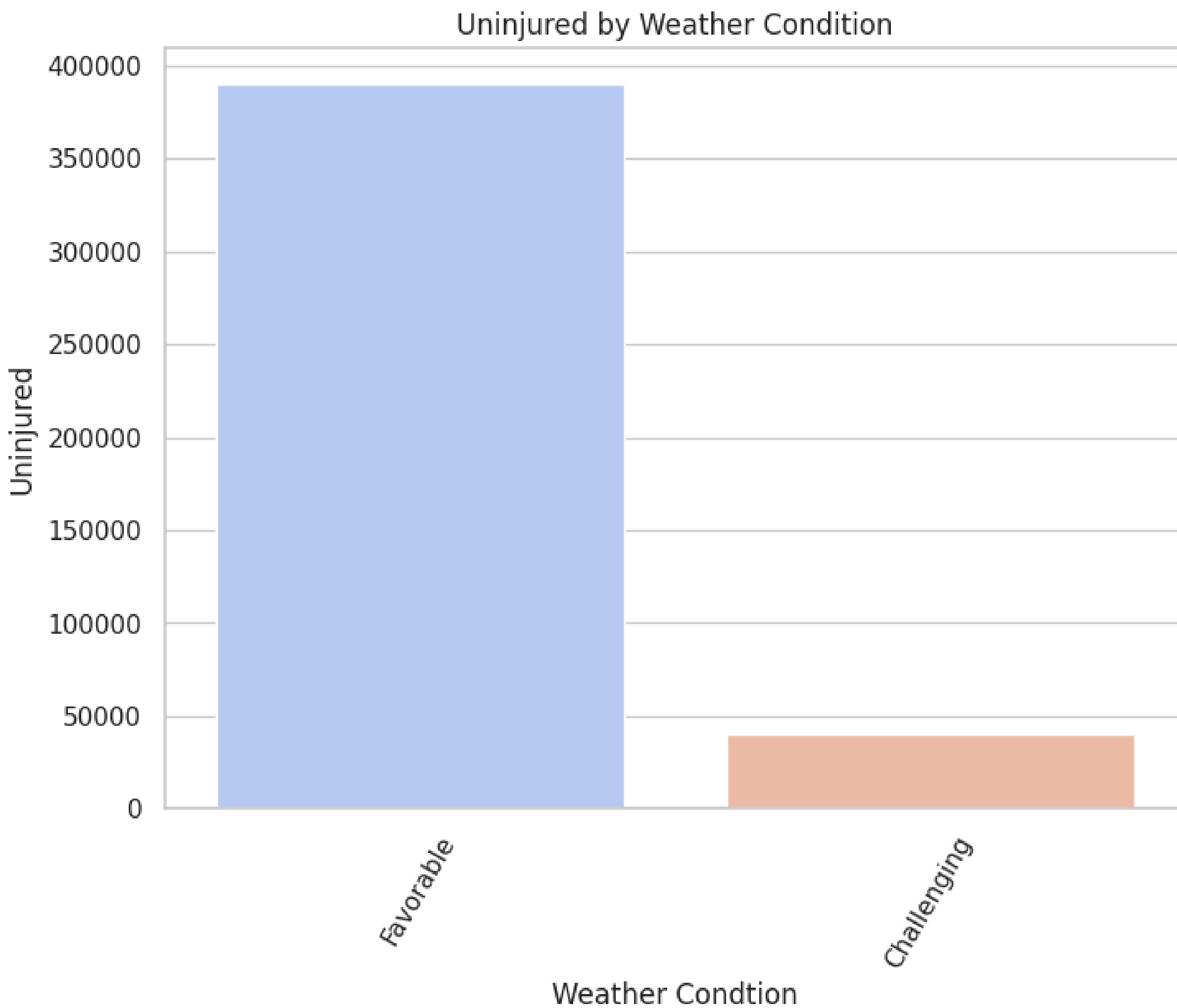
In [65]:
```
# Fatal Injuries Vs Weather Conditions
weather_fatal_injuries = df.groupby('weather_category')['total_fatal_injuries'].sum
plt.figure(figsize=(8,6))
sns.barplot(x=weather_fatal_injuries.index,y=weather_fatal_injuries.values,palette=
plt.title('Fatal Injuries by Weather Conditions')
plt.xlabel('Weather Condition')
plt.ylabel('Fatal Injuries')
plt.xticks(rotation=60)
plt.show()
```



Observations:

- "VMC" (Favorable) weather conditions are also associated with a higher number of fatal injuries compared to "IMC".

In [66]:
```
# Uninjured Vs Weather Condition
weather_uninjured = df.groupby('weather_category')['total_uninjured'].sum().sort_va
plt.figure(figsize=(8,6))
sns.barplot(x=weather_uninjured.index,y=weather_uninjured.values,palette='coolwarm'
plt.title('Uninjured by Weather Condition')
plt.xlabel('Weather Condtion')
plt.ylabel('Uninjured')
plt.xticks(rotation=60)
plt.show()
```

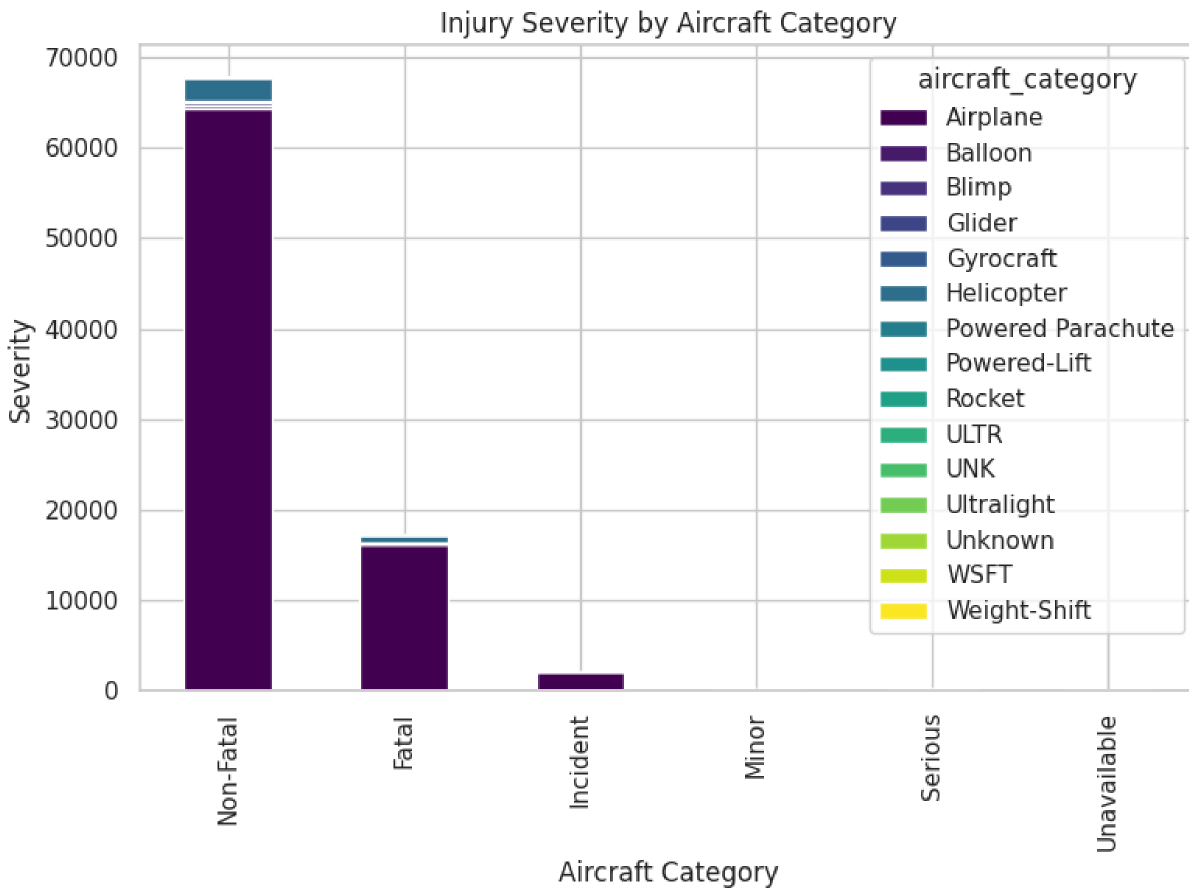


Observations:

- "VMC" weather conditions show a higher number of uninjured individuals, reflecting the overall higher number of accidents in these conditions.

Categorical Vs Categorical

In [67]:
```
# Injury severity Vs Aircraft Category
cross_tab = pd.crosstab(df['injury_severity'],df['aircraft_category'])

#Sort by total count
cross_tab['total']= cross_tab.sum(axis=1)
cross_tab=cross_tab.sort_values(by='total',ascending=False).drop(columns=['total'])
cross_tab.plot(kind='bar', stacked=True, colormap='viridis', figsize=(8, 6))
plt.xlabel('Aircraft Category')
plt.ylabel('Severity')
plt.title('Injury Severity by Aircraft Category')
plt.tight_layout()
plt.show()
```

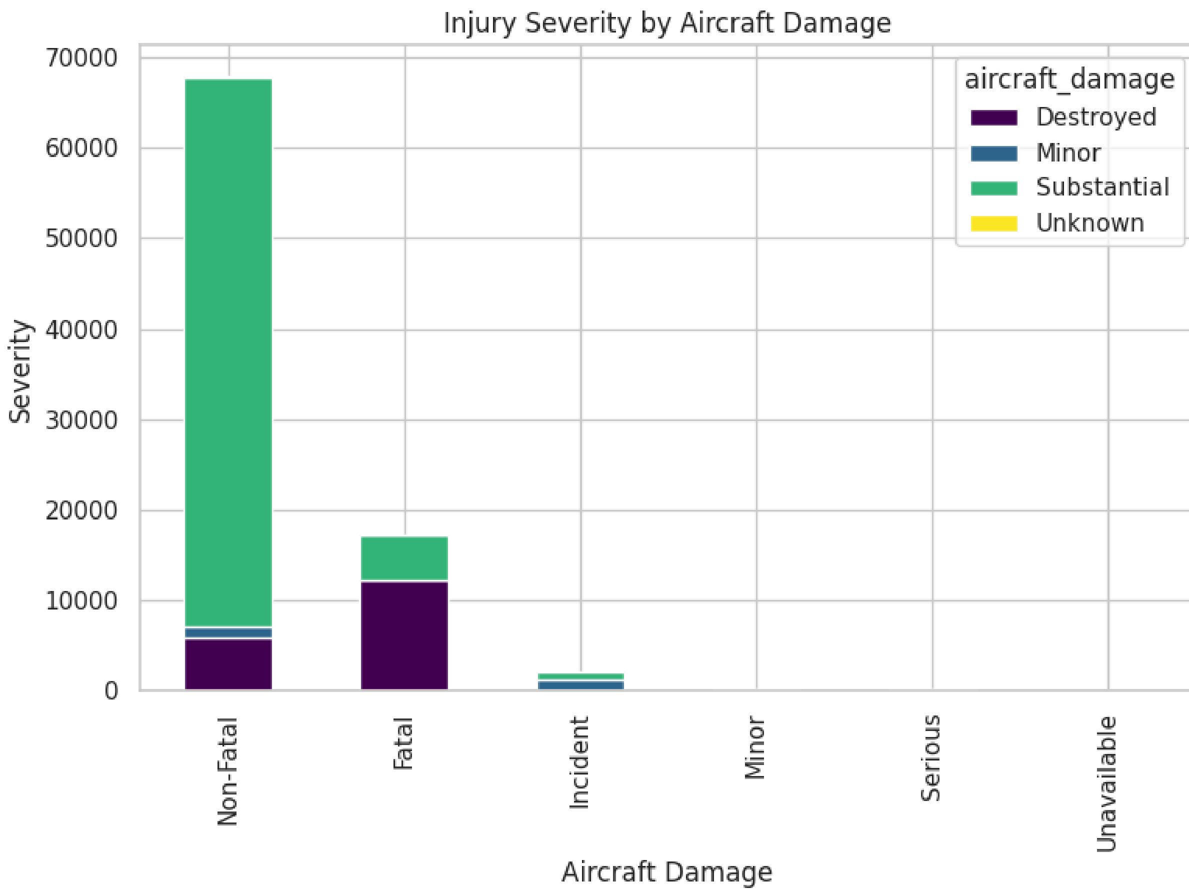


Observations:

- Across all aircraft categories, "Non-Fatal" and "Fatal" are the most common injury severities reported.
- "Airplane" category contributes the most to each severity level due to the sheer number of accidents involving this type.
- While smaller in count, "Helicopter" and "Glider" accidents also result in a mix of injury severities.

In [68]:

```
# Injury severity Vs Aircraft Category
cross_tab = pd.crosstab(df['injury_severity'],df['aircraft_damage'])

#Sort by total count
cross_tab['total']= cross_tab.sum(axis=1)
cross_tab=cross_tab.sort_values(by='total',ascending=False).drop(columns=['total'])
cross_tab.plot(kind='bar', stacked=True, colormap='viridis', figsize=(8, 6))
plt.xlabel('Aircraft Damage')
plt.ylabel('Severity')
plt.title('Injury Severity by Aircraft Damage')
plt.tight_layout()
plt.show()
```

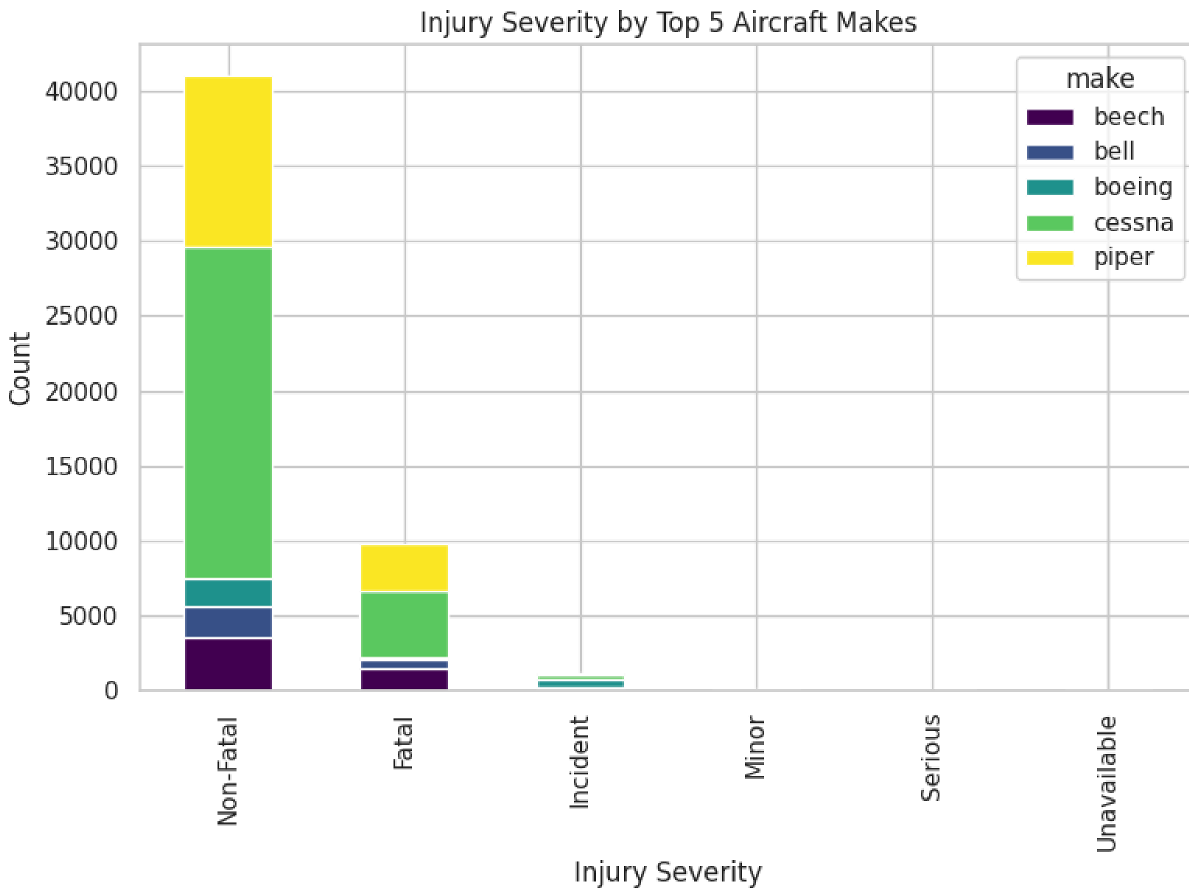


Observations:

- "Destroyed" aircraft are overwhelmingly associated with "Fatal" injuries.
- "Substantial" damage is associated with all injury severity levels, but with a higher proportion of "Non-Fatal" and "Fatal".
- "Minor" damage is primarily associated with "Non-Fatal" accidents.

In [69]:
```
#Injury severity vs make
# Get top 5 makes
top_makes = df['make'].value_counts().head(5).index

# Filter the DataFrame to include only top 5 makes
df_top5_make = df[df['make'].isin(top_makes)]

# Create a crosstab between injury severity and make
cross_tab = pd.crosstab(df_top5_make['injury_severity'], df_top5_make['make'])

# Sort by total row-wise count
cross_tab['total'] = cross_tab.sum(axis=1)
cross_tab = cross_tab.sort_values(by='total', ascending=False).drop(columns=['total

# Plot
cross_tab.plot(kind='bar', stacked=True, colormap='viridis', figsize=(8, 6))
plt.xlabel('Injury Severity')
plt.ylabel('Count')
```

```
plt.title('Injury Severity by Top 5 Aircraft Makes')
plt.tight_layout()
plt.show()
```



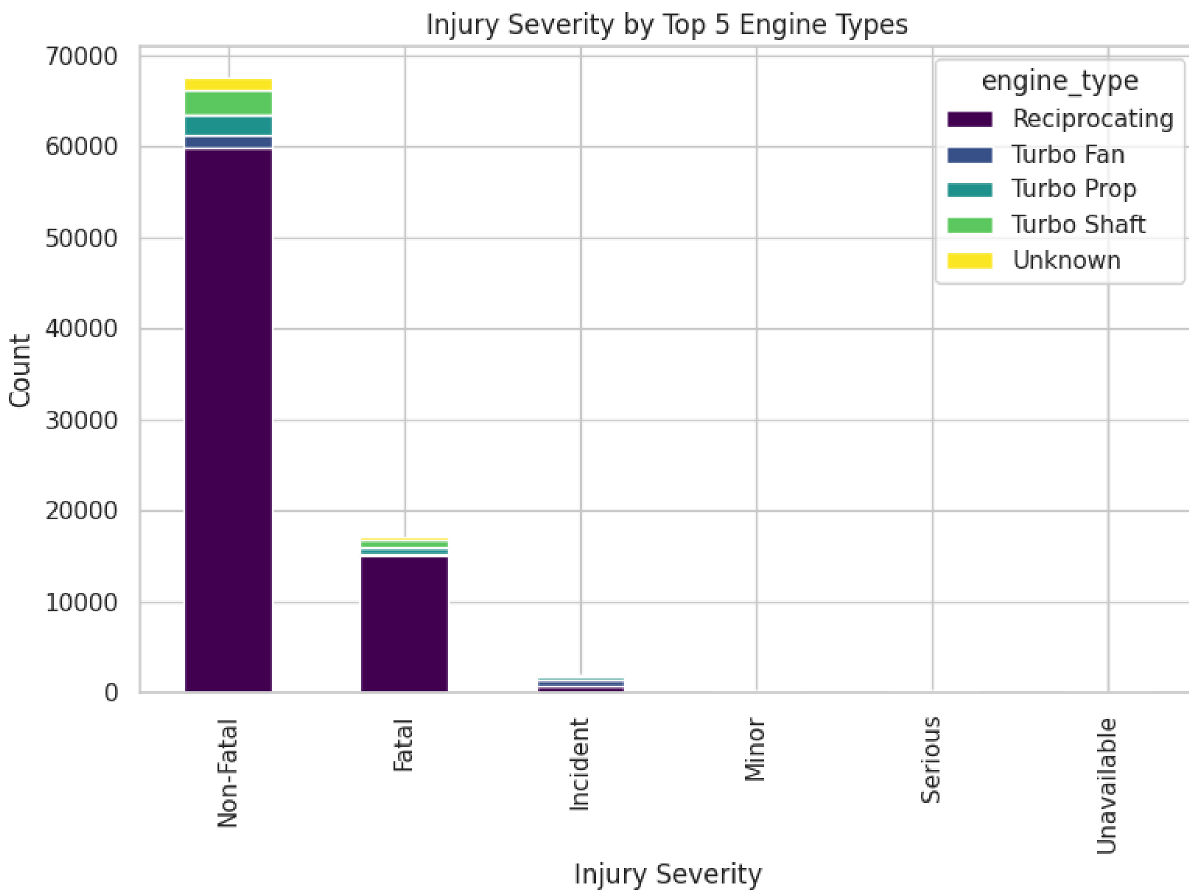
Observations:

- For the top 5 makes, "Non-Fatal" and "Fatal" are the most frequent outcomes.
- "Boeing" and "Airbus" have the highest counts across all severity levels, particularly for fatal accidents, likely reflecting their role in commercial aviation.
- "Cessna" and "Piper" have high counts of "Non-Fatal" accidents, aligning with their use in general aviation.

In [70]:

```
#Injury severity vs engine type
# Get top 5
top_engines = df['engine_type'].value_counts().head(5).index

# Filter the DataFrame to include only top 5 makes
df_top5_engines = df[df['engine_type'].isin(top_engines)]

# Create a crosstab between injury severity and make
cross_tab = pd.crosstab(df_top5_engines['injury_severity'], df_top5_engines['engine

# Sort by total row-wise count
cross_tab['total'] = cross_tab.sum(axis=1)
cross_tab = cross_tab.sort_values(by='total', ascending=False).drop(columns=['total
```

```
# Plot
cross_tab.plot(kind='bar', stacked=True, colormap='viridis', figsize=(8, 6))
plt.xlabel('Injury Severity')
plt.ylabel('Count')
plt.title('Injury Severity by Top 5 Engine Types')
plt.tight_layout()
plt.show()
```

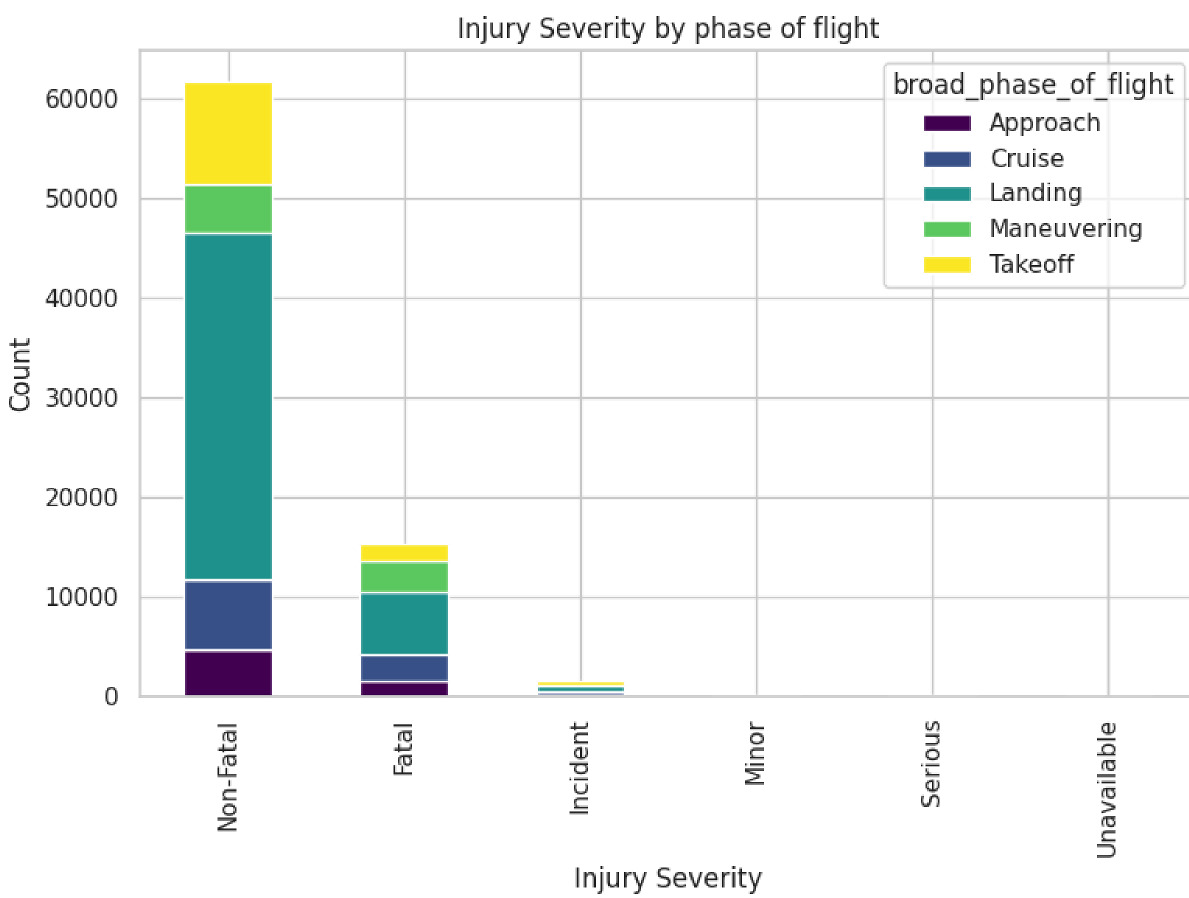


Observations:

- Across the top 5 engine types, "Non-Fatal" and "Fatal" are the dominant injury severities.
- "Jet" engines contribute the most to "Fatal" injuries, consistent with commercial airline accidents.
- "Reciprocating" engines contribute significantly to "Non-Fatal" and
- "Fatal" accidents, reflecting the large number of incidents involving this engine type.

In [71]:
```
#Injury severity vs phase of flight
# Get top 5
top_phases = df['broad_phase_of_flight'].value_counts().head(5).index

# Filter the DataFrame to include only top 5 makes
```

```
df_top5_phase = df[df['broad_phase_of_flight'].isin(top_phases)]

# Create a crosstab between injury severity and make
cross_tab = pd.crosstab(df_top5_phase['injury_severity'], df_top5_phase['broad_phas

# Sort by total row-wise count
cross_tab['total'] = cross_tab.sum(axis=1)
cross_tab = cross_tab.sort_values(by='total', ascending=False).drop(columns=['total

# Plot
cross_tab.plot(kind='bar', stacked=True, colormap='viridis', figsize=(8, 6))
plt.xlabel('Injury Severity')
plt.ylabel('Count')
plt.title('Injury Severity by phase of flight')
plt.tight_layout()
plt.show()
```

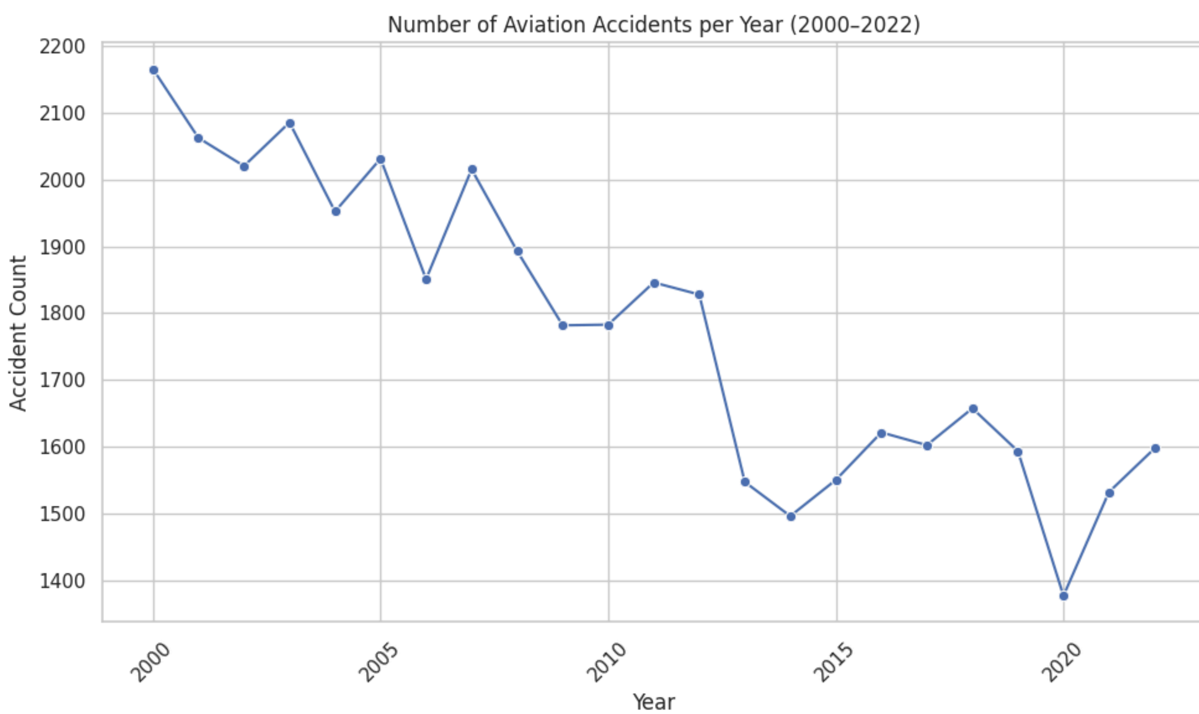


Observations:

- The "Landing", "Takeoff", and "Approach" phases have the highest counts of accidents across all injury severities.
- The "Cruise" phase, while having fewer total accidents, shows a significant proportion of "Fatal" outcomes when they do occur.

Trends over time

In [72]:
```
 #Filter data between 2000 and 2022
df_yearly = df[(df['year'] >= 2000) & (df['year'] <= 2022)]

# Count number of accidents per year
accident_counts = df_yearly['year'].value_counts().sort_index().reset_index()
accident_counts.columns = ['year', 'accident_count']

# Plot
plt.figure(figsize=(10, 6))
sns.lineplot(data=accident_counts, x='year', y='accident_count', marker='o')
plt.title('Number of Aviation Accidents per Year (2000–2022)')
plt.xlabel('Year')
plt.ylabel('Accident Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Observations:

- There has been a general downward trend in the number of aviation accidents per year from 2000 to 2022.
- There are some fluctuations year-to-year, but the overall pattern shows a decrease in accident frequency over this period.

In [73]:
```
# Filter data between 2000 and 2020
df_yearly = df[(df['year'] >= 2000) & (df['year'] <= 2022)]

# Group by year and sum total injuries
```

```
yearly_injuries = df_yearly.groupby('year')['total_injuries'].sum().reset_index()

# Plot
plt.figure(figsize=(10, 6))
sns.lineplot(data=yearly_injuries, x='year', y='total_injuries', marker='o')
plt.title('Total Injuries from Aviation Accidents (2000-2022)')
plt.xlabel('Year')
plt.ylabel('Total Injuries')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

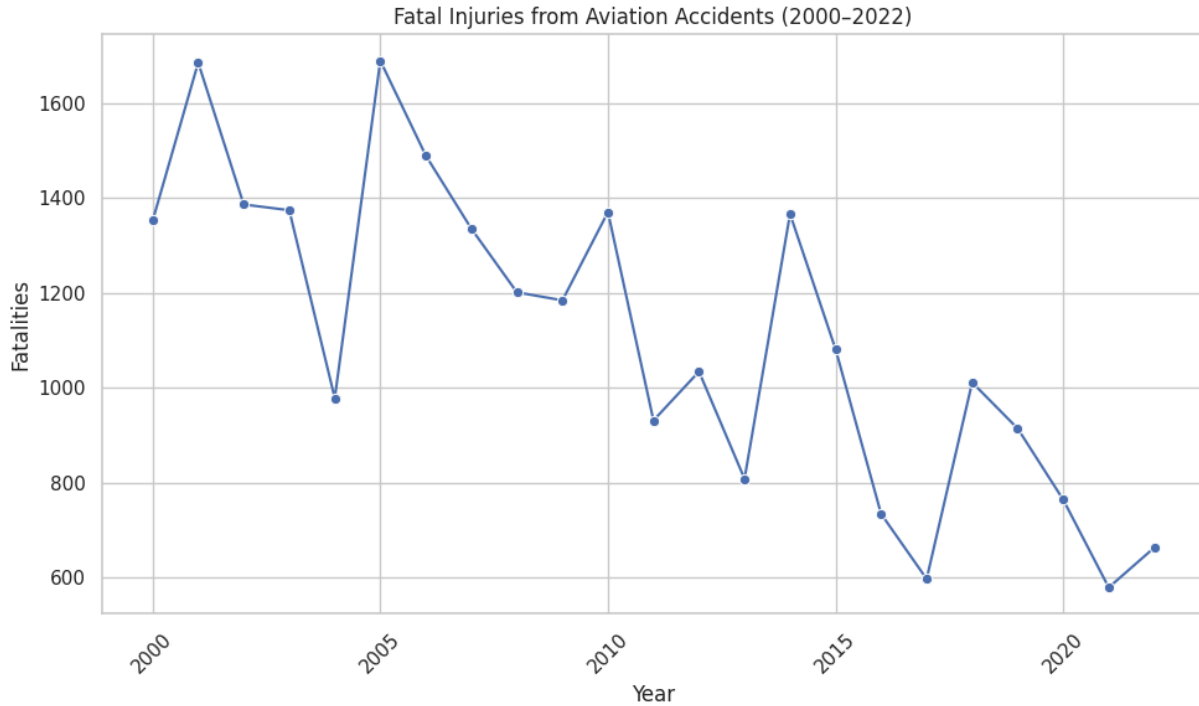
Observations:

- Similar to the accident count, there is a general downward trend in the total number of injuries from aviation accidents between 2000 and 2022.
- This suggests that not only are accidents becoming less frequent, but the total human impact in terms of injuries is also decreasing

In [74]:

```
# Filter data between 2000 and 2020
df_yearly = df[(df['year'] >= 2000) & (df['year'] <= 2022)]

# Group by year and sum total injuries
yearly_injuries = df_yearly.groupby('year')['total_fatal_injuries'].sum().reset_ind

# Plot
plt.figure(figsize=(10, 6))
sns.lineplot(data=yearly_injuries, x='year', y='total_fatal_injuries', marker='o')
plt.title('Fatal Injuries from Aviation Accidents (2000-2022)')
plt.xlabel('Year')
plt.ylabel('Fatalities')
```

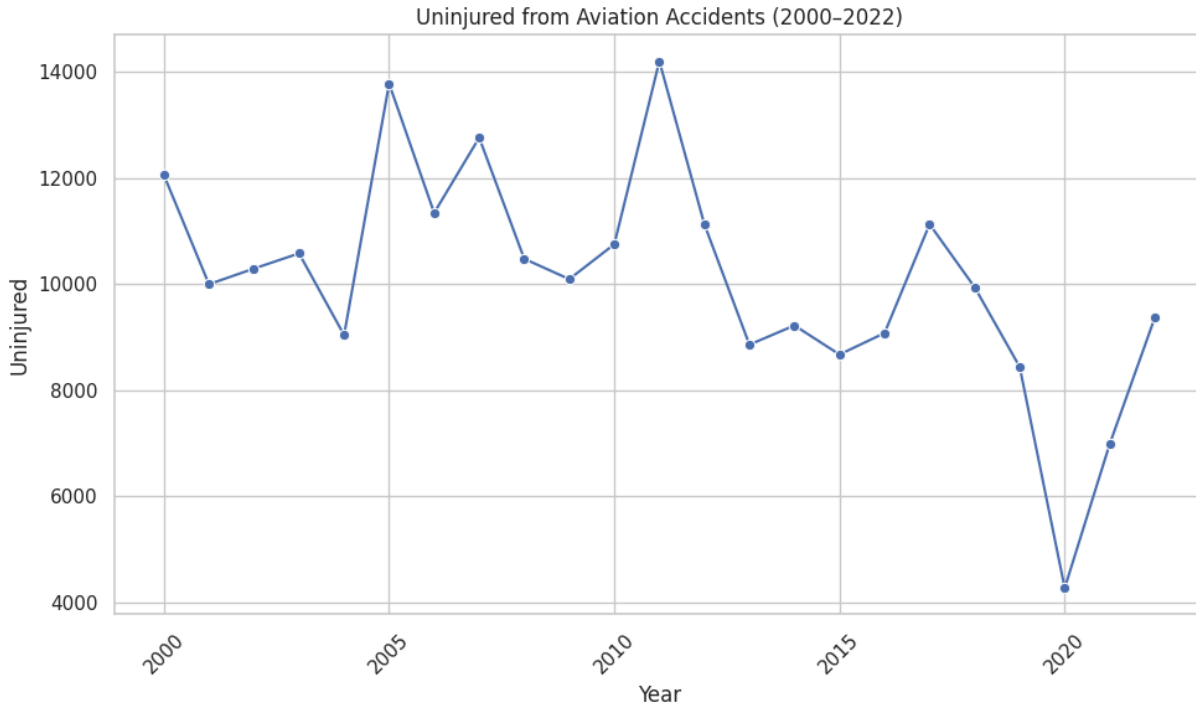
```
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Observations:

- The number of fatal injuries from aviation accidents has also generally decreased from 2000 to 2022.
- There can be significant year-to-year variability, often influenced by a small number of high-fatality events.

In [75]:
```
# Filter data between 2000 and 2020
df_yearly = df[(df['year'] >= 2000) & (df['year'] <= 2022)]

# Group by year and sum total injuries
yearly_injuries = df_yearly.groupby('year')['total_uninjured'].sum().reset_index()

# Plot
plt.figure(figsize=(10, 6))
sns.lineplot(data=yearly_injuries, x='year', y='total_uninjured', marker='o')
plt.title('Uninjured from Aviation Accidents (2000-2022)')
plt.xlabel('Year')
plt.ylabel('Uninjured')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

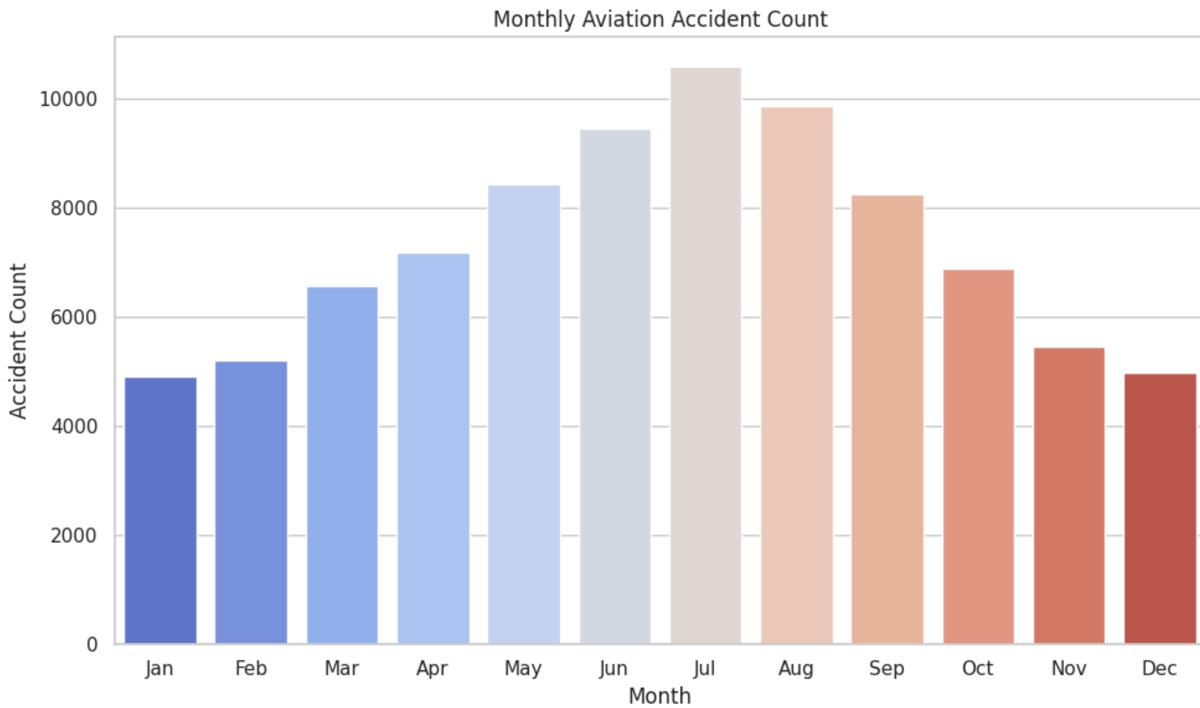


Observations:

- The trend in the number of uninjured individuals involved in accidents appears to follow a similar pattern to total accidents and injuries, showing a general decrease over the years.

In [76]:
```
# Count accidents by month
monthly_accidents = df['month'].value_counts().sort_index().reset_index()
monthly_accidents.columns = ['month', 'accident_count']

# Map numeric months to names
month_map = {
    1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr',
    5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug',
    9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'
}
monthly_accidents['month_name'] = monthly_accidents['month'].map(month_map)

# Ensure correct month order
monthly_accidents['month_name'] = pd.Categorical(
    monthly_accidents['month_name'],
    categories=list(month_map.values()),
    ordered=True
)

# Sort by month order
monthly_accidents = monthly_accidents.sort_values('month_name')

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(data=monthly_accidents, x='month_name', y='accident_count', palette='co
plt.title('Monthly Aviation Accident Count')
```

```
plt.xlabel('Month')
plt.ylabel('Accident Count')
plt.tight_layout()
plt.show()
```

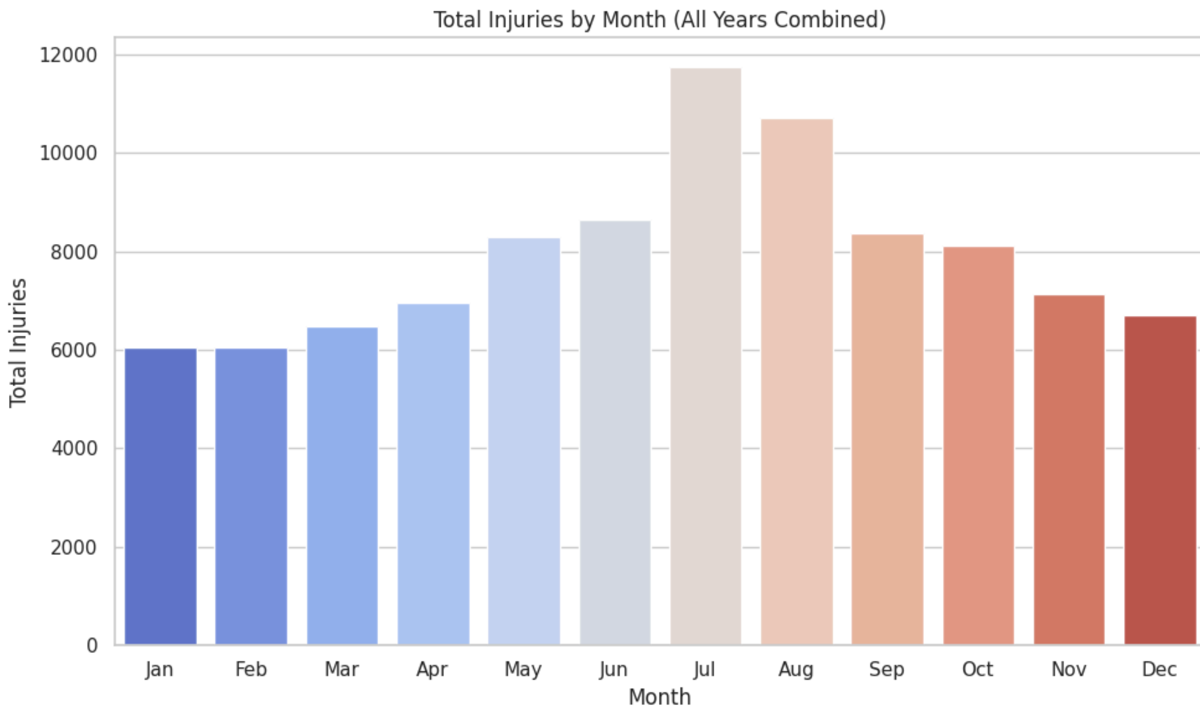


Observations:

- The number of aviation accidents shows a seasonal pattern.
- The summer months (June, July, August) tend to have a higher number of accidents compared to other months.
- This could be related to increased flight activity during these months or weather patterns.

In [77]:
```
# Group by month and sum total injuries
monthly_injuries = df.groupby('month')['total_injuries'].sum().reset_index()

# Map month numbers to names
month_map = {
    1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr',
    5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug',
    9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'
}
monthly_injuries['month_name'] = monthly_injuries['month'].map(month_map)

# Sort by calendar order
monthly_injuries['month_name'] = pd.Categorical(
    monthly_injuries['month_name'],
    categories=list(month_map.values()),
    ordered=True
)
```

```
monthly_injuries = monthly_injuries.sort_values('month_name')

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(data=monthly_injuries, x='month_name', y='total_injuries', palette='coo
plt.title('Total Injuries by Month (All Years Combined)')
plt.xlabel('Month')
plt.ylabel('Total Injuries')
plt.tight_layout()
plt.show()
```

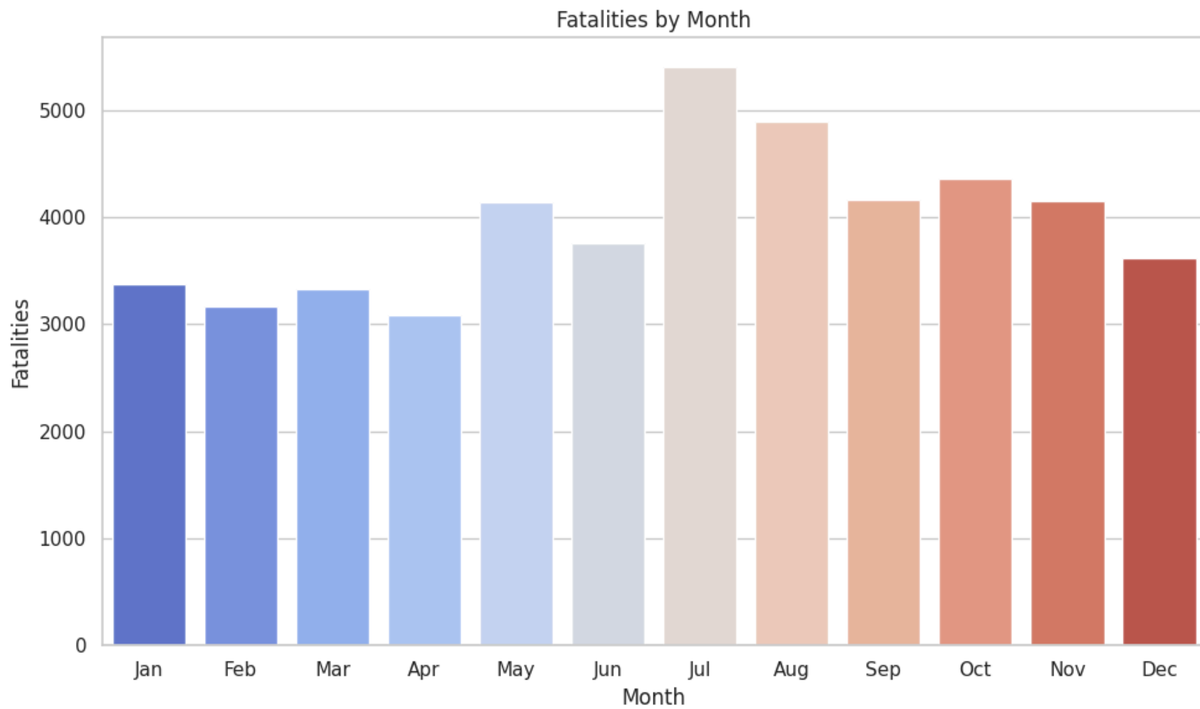


Observations:

- The total number of injuries also shows a seasonal trend, peaking in the summer months, aligning with the accident count.

In [78]:
```
# Group by month and sum fatal injuries
monthly_fatal_injuries = df.groupby('month')['total_fatal_injuries'].sum().reset_in

# Map month numbers to names
month_map = {
    1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr',
    5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug',
    9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'
}
monthly_fatal_injuries['month_name'] = monthly_fatal_injuries['month'].map(month_ma

# Sort by calendar order
monthly_fatal_injuries['month_name'] = pd.Categorical(
    monthly_fatal_injuries['month_name'],
    categories=list(month_map.values()),
    ordered=True
)
```

```
monthly_fatal_injuries = monthly_fatal_injuries.sort_values('month_name')

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(data=monthly_fatal_injuries, x='month_name', y='total_fatal_injuries',
plt.title('Fatalities by Month')
plt.xlabel('Month')
plt.ylabel('Fatalities')
plt.tight_layout()
plt.show()
```

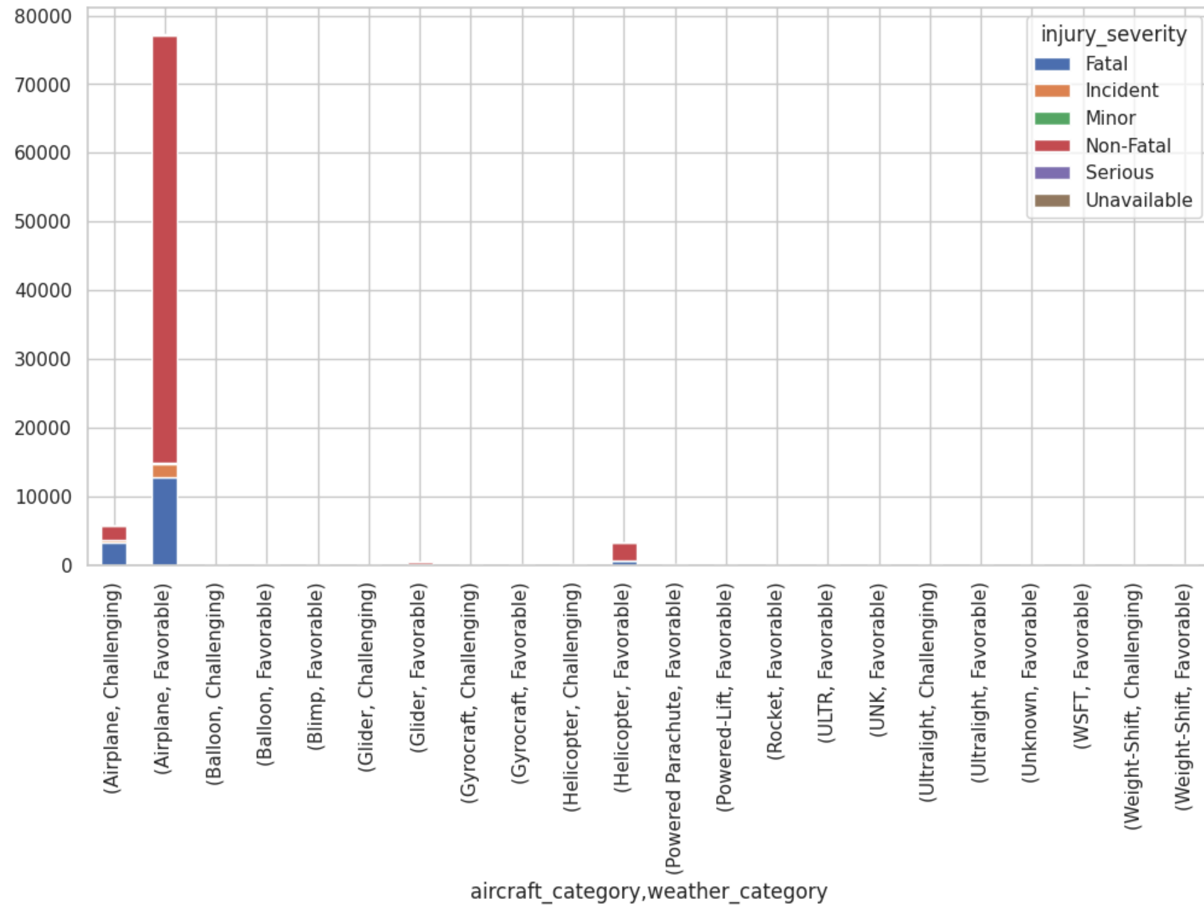


Observations:

- The distribution of fatalities by month also tends to be higher in the summer months, following the overall accident and injury trends.

Multi-variate Analysis

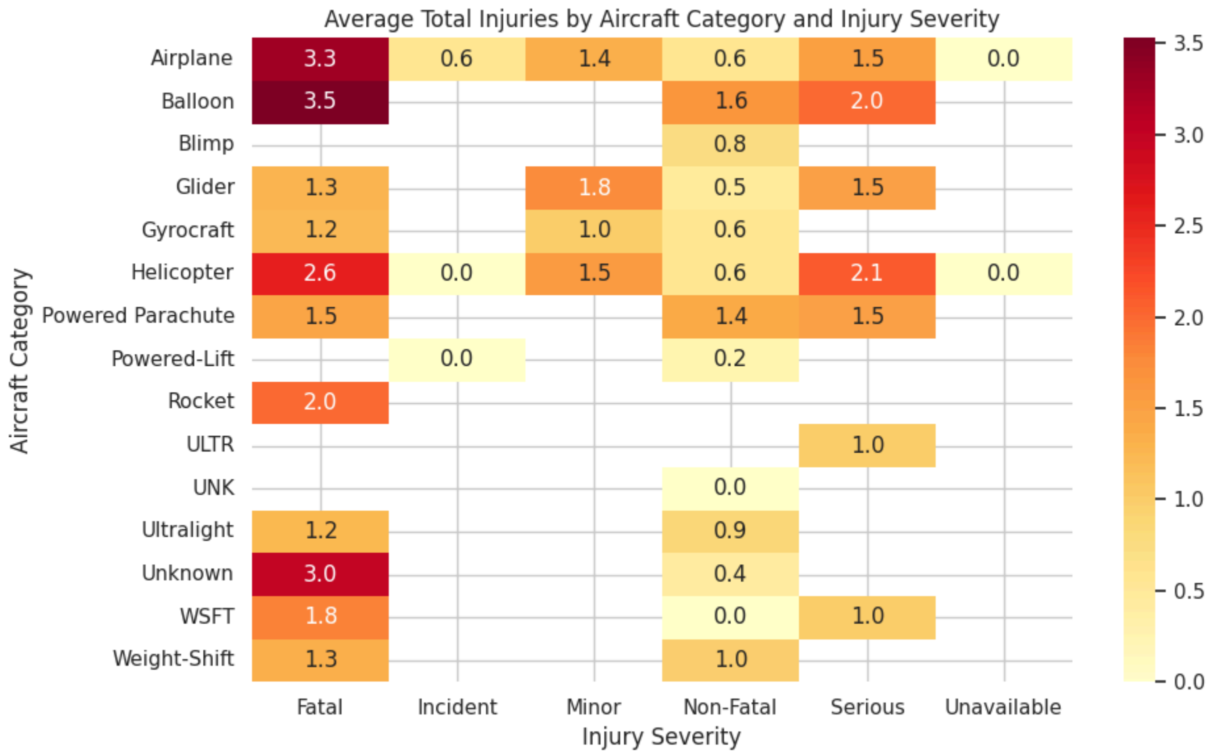
In [79]:
```
pd.crosstab([df['aircraft_category'], df['weather_category']],
            df['injury_severity']).plot(kind='bar', stacked=True, figsize=(12,6));
```



Observations:

For "Fatal" accidents, "Airplane" category shows the highest average number of total injuries. For "Serious" and "Minor" severities, the average number of total injuries is much lower across all aircraft categories. The heatmap clearly shows that "Fatal" outcomes in "Airplane" accidents have the highest average human impact.

In [80]:
```
pivot_data = df.pivot_table(
    index='aircraft_category',
    columns='injury_severity',
    values='total_injuries',
    aggfunc='mean'
)

plt.figure(figsize=(10, 6))
sns.heatmap(pivot_data, annot=True, fmt=".1f", cmap="YlOrRd")
plt.title("Average Total Injuries by Aircraft Category and Injury Severity")
plt.xlabel("Injury Severity")
plt.ylabel("Aircraft Category")
plt.tight_layout()
plt.show()
```



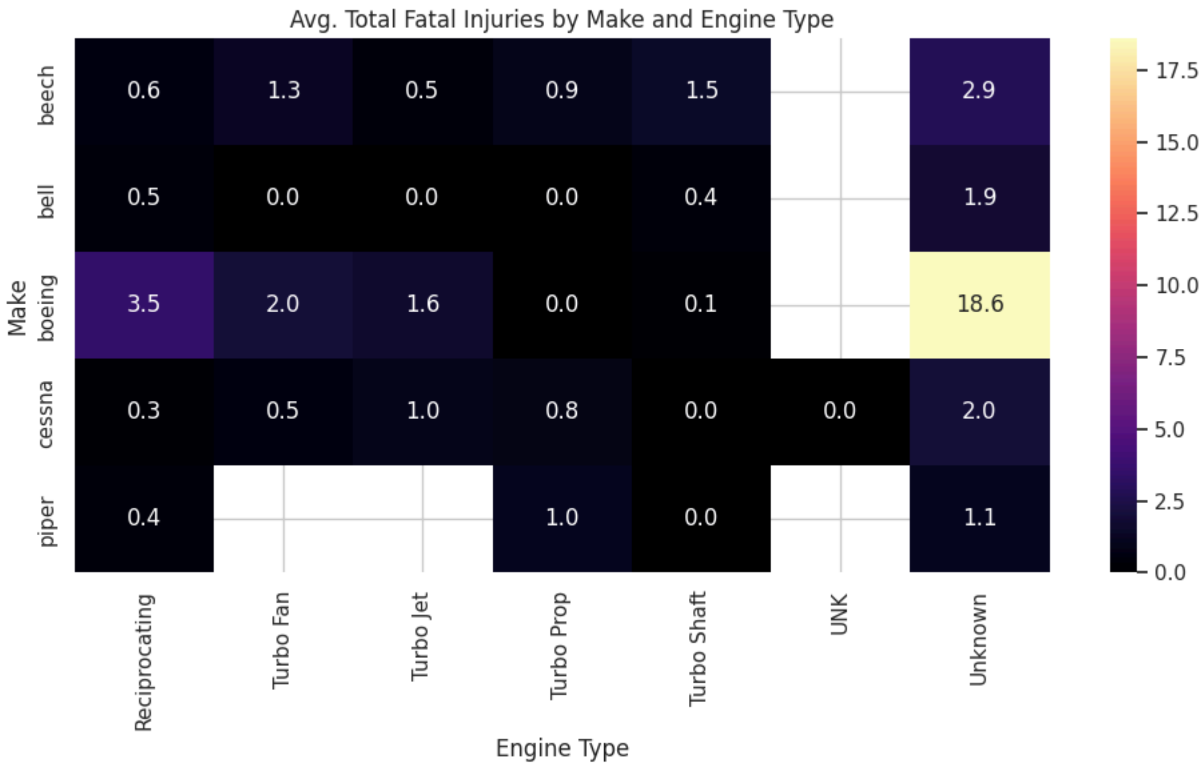
Observations:

- Among the top 5 makes, "Boeing" aircraft with "Jet" engines show the highest average number of fatal injuries.
- "Airbus" with "Jet" engines also shows a high average number of fatal injuries.

Makes like "Cessna" and "Piper" with

- "Reciprocating" engines have much lower average fatal injuries per accident, even though they have high accident counts.

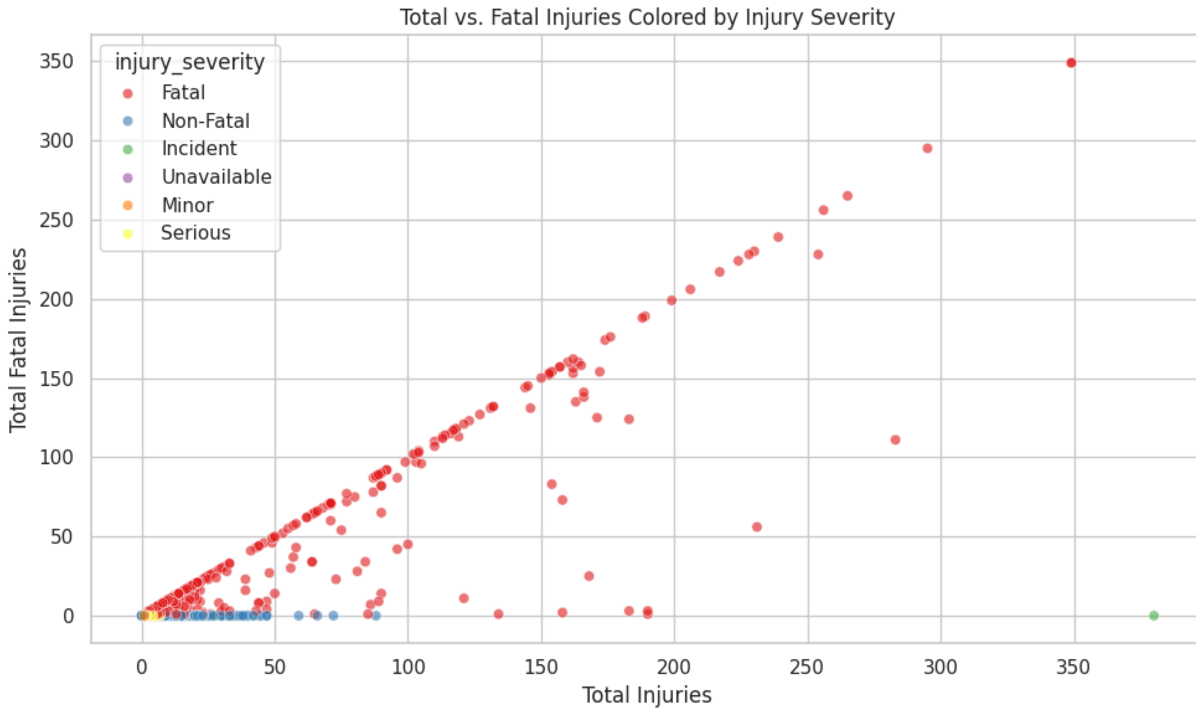
In [81]:
```
top_makes = df['make'].value_counts().head(5).index
filtered_df = df[df['make'].isin(top_makes)]

pivot_data = filtered_df.pivot_table(
    index='make',
    columns='engine_type',
    values='total_fatal_injuries',
    aggfunc='mean'
)

plt.figure(figsize=(10, 6))
sns.heatmap(pivot_data, annot=True, fmt=".1f", cmap="magma")
plt.title("Avg. Total Fatal Injuries by Make and Engine Type")
plt.xlabel("Engine Type")
plt.ylabel("Make")
plt.tight_layout()
plt.show()
```



bservations:

- Points colored "Fatal" are generally concentrated in the upper right portion of the scatter plot, where both total injuries and fatal injuries are high.
- Points colored "Non-Fatal" are clustered along the x-axis, indicating accidents with total injuries but zero fatal injuries.
- Points colored "Serious" and "Minor" are found in between, with lower total injuries and a range of fatal injury counts (mostly low). This visual reinforces the relationship between the different injury metrics and the categorized injury severity.

In [82]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=df,
    x='total_injuries',
    y='total_fatal_injuries',
    hue='injury_severity',
    alpha=0.6,
    palette='Set1'
)
plt.title("Total vs. Fatal Injuries Colored by Injury Severity")
plt.xlabel("Total Injuries")
plt.ylabel("Total Fatal Injuries")
plt.tight_layout()
plt.show()
```



Observations:

- Points colored "Fatal" are generally concentrated in the upper right portion of the scatter plot, where both total injuries and fatal injuries are high.
- Points colored "Non-Fatal" are clustered along the x-axis, indicating accidents with total injuries but zero fatal injuries.
- Points colored "Serious" and "Minor" are found in between, with lower total injuries and a range of fatal injury counts (mostly low). This visual reinforces the relationship between the different injury metrics and the categorized injury severity.

Data Preprocessing

In [83]:
```
df['injury_severity_category'].value_counts()
```

Out[83]:

injury_severity_category	count
No Injuries	48019
Isolated Injury	20322
Few Injuries	17412
Moderate Injuries	1499
Mass Casualties	519

dtype: int64

In [84]:

```
df.nunique().sort_values(ascending=False)
```

Out[84]:

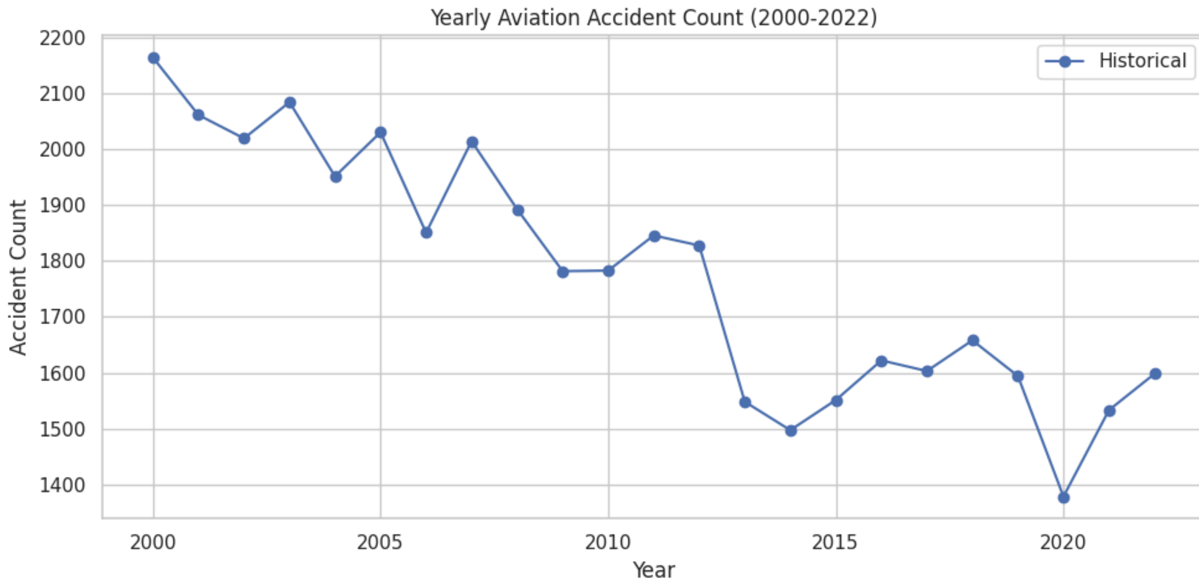
	0
location	27324
event_date	14767
model	12212
make	7559
total_uninjured	377
country	215
total_injuries	136
total_fatal_injuries	118
total_minor_injuries	54
total_serious_injuries	49
year	45
purpose_of_flight	26
aircraft_category	15
month	12
engine_type	12
broad_phase_of_flight	12
number_of_engines	7
day_of_week	7
injury_severity	6
fatality_category	5
injury_severity_category	5
aircraft_damage	4
quarter	4
amateur_built	2
investigation_type	2
weather_condition	2
weather_category	2

dtype: int64

In [91]:

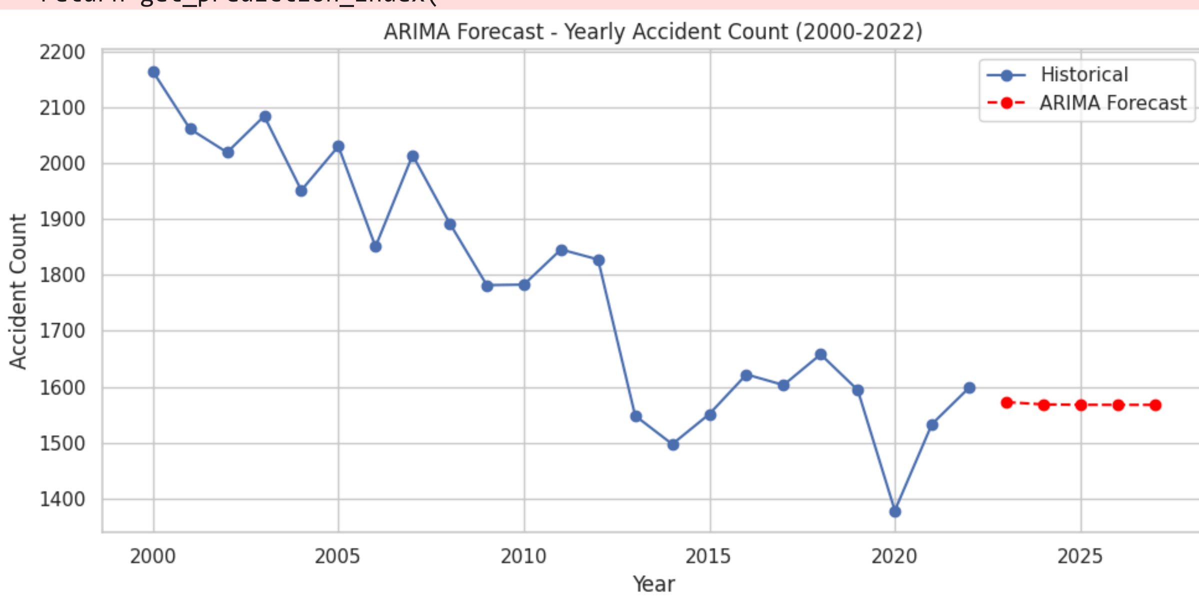
```
import pandas as pd
import matplotlib.pyplot as plt
```

```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Ensure event_date is in datetime format
df['event_date'] = pd.to_datetime(df['event_date'], errors='coerce')

# Filter data between 2000 and 2022
df_filtered = df[(df['event_date'].dt.year >= 2000) & (df['event_date'].dt.year <=

# Aggregate yearly accident counts
yearly_accidents = df_filtered.set_index('event_date').resample('Y').size()
# Convert index to year for clarity
yearly_accidents.index = yearly_accidents.index.year

# Plot historical data
plt.figure(figsize=(10,5))
plt.plot(yearly_accidents.index, yearly_accidents.values, marker='o', label='Histor
plt.title("Yearly Aviation Accident Count (2000-2022)")
plt.xlabel("Year")
plt.ylabel("Accident Count")
plt.legend()
plt.tight_layout()
plt.show()

# --- ARIMA Forecasting ---

# Check stationarity using ADF test (for your reference)
adf_result = adfuller(yearly_accidents)
print("ARIMA: ADF Statistic:", adf_result[0], "p-value:", adf_result[1])

# Fit ARIMA. We use order=(1,1,1) as an example. Adjust p,d,q based on ACF/PACF if
arima_model = ARIMA(yearly_accidents, order=(1,1,1))
arima_result = arima_model.fit()

# Forecast next 5 years (e.g., 2023-2027)
forecast_arima = arima_result.forecast(steps=5)

# Plot ARIMA forecast
plt.figure(figsize=(10,5))
plt.plot(yearly_accidents.index, yearly_accidents.values, marker='o', label='Histor
plt.plot(range(2023, 2023+5), forecast_arima, marker='o', color='red', linestyle='-
plt.title("ARIMA Forecast - Yearly Accident Count (2000-2022)")
plt.xlabel("Year")
plt.ylabel("Accident Count")
plt.legend()
plt.tight_layout()
plt.show()
```




```
ARIMA: ADF Statistic: 1.1368862073445611 p-value: 0.9955194564401095
```

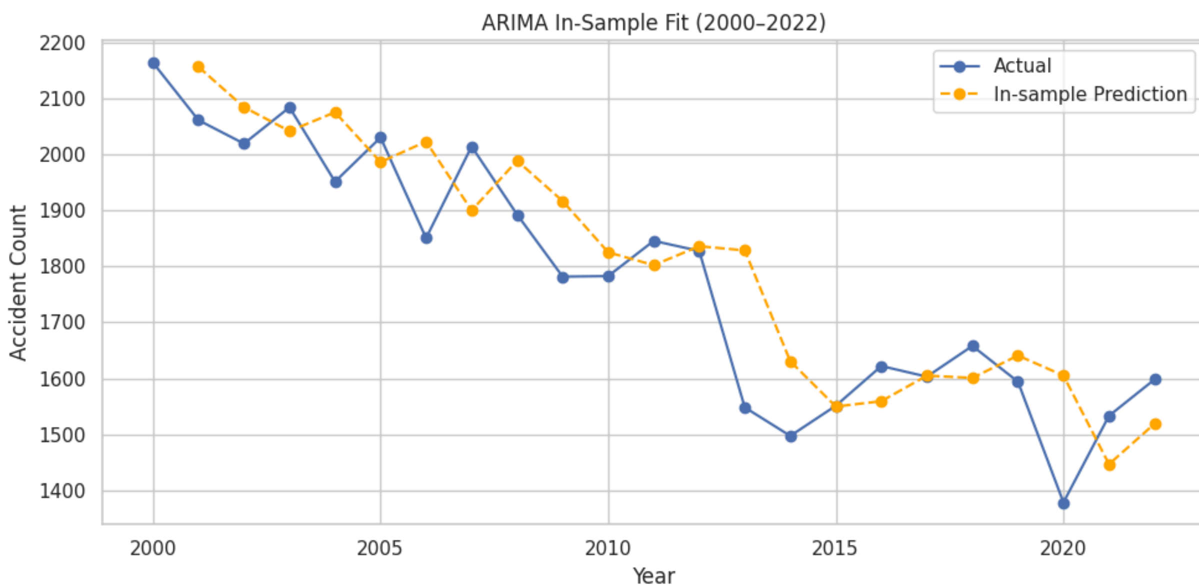
```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning: An unsupported index was provided. As a result, forecasts cannot be generat
ed. To use the model for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning: An unsupported index was provided. As a result, forecasts cannot be generat
ed. To use the model for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Value
Warning: An unsupported index was provided. As a result, forecasts cannot be generat
ed. To use the model for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:837: Value
Warning: No supported index is available. Prediction results will be given with an i
nteger index beginning at `start`.
  return get_prediction_index(
```



In [92]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np
```

```
# In-sample prediction
in_sample_pred = arima_result.predict(start=1, end=len(yearly_accidents)-1, typ='le
actual_values = yearly_accidents[1:]

# Evaluation metrics
mae = mean_absolute_error(actual_values, in_sample_pred)
rmse = np.sqrt(mean_squared_error(actual_values, in_sample_pred))

print("\n=== ARIMA In-Sample Evaluation ===")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

# Plot actual vs predicted for in-sample
plt.figure(figsize=(10,5))
plt.plot(yearly_accidents.index, yearly_accidents.values, marker='o', label='Actual
plt.plot(yearly_accidents.index[1:], in_sample_pred, marker='o', linestyle='--', co
plt.title("ARIMA In-Sample Fit (2000–2022)")
plt.xlabel("Year")
plt.ylabel("Accident Count")
plt.legend()
plt.tight_layout()
plt.show()
```

```
=== ARIMA In-Sample Evaluation ===
Mean Absolute Error (MAE): 89.21
Root Mean Squared Error (RMSE): 112.48
```



Insights and Observations

1. **Overall Trend Capture:** The red line representing the ARIMA predictions generally follows the downward trend observed in the actual test data (orange line). This suggests the model is capturing the underlying trend of decreasing accident counts.

2. **Fluctuation Handling:** The ARIMA model seems to smooth out the year-to-year fluctuations present in the actual data. The predicted line is less volatile than the actual data points in the test set. This is typical of many time series models that aim to capture

the underlying pattern rather than the specific noise or irregularities in individual periods.

3. **Accuracy in the Test Period:** To make more precise observations about accuracy, you would need to compare the predicted values (red line) to the actual values (orange line) for each year in the test period. Visually, you can see how close the red dots are to the orange dots. The RMSE and MAE values printed below the plot provide quantitative measures of this accuracy. Lower values indicate better performance.

4. **Potential for Improvement:** While the model captures the trend, there might be noticeable differences between the predicted and actual values in certain years of the test set. This suggests there's room for improvement in the model's ability to capture the nuances or specific deviations in the time series. This could involve:

5. **Tuning the ARIMA parameters (p, d, q).**

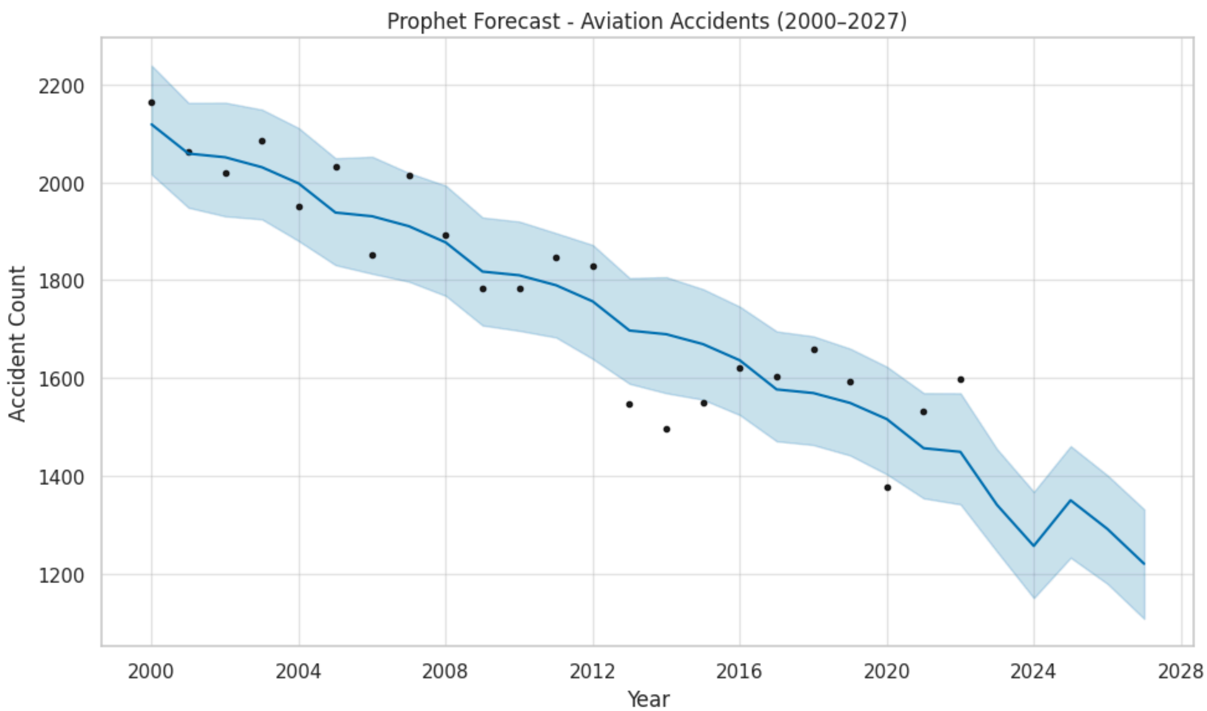
Exploring other time series models (e.g., Prophet, seasonal ARIMA). Considering external factors that might influence accident rates (though this would require more complex modeling). Forecasting Beyond the Test Period: The plot shows predictions only up to the end of the test period (2022). If you were to forecast further into the future, the model would continue the trend it has learned. The reliability of those future forecasts would depend on how well the historical trend continues and whether any significant external events occur.

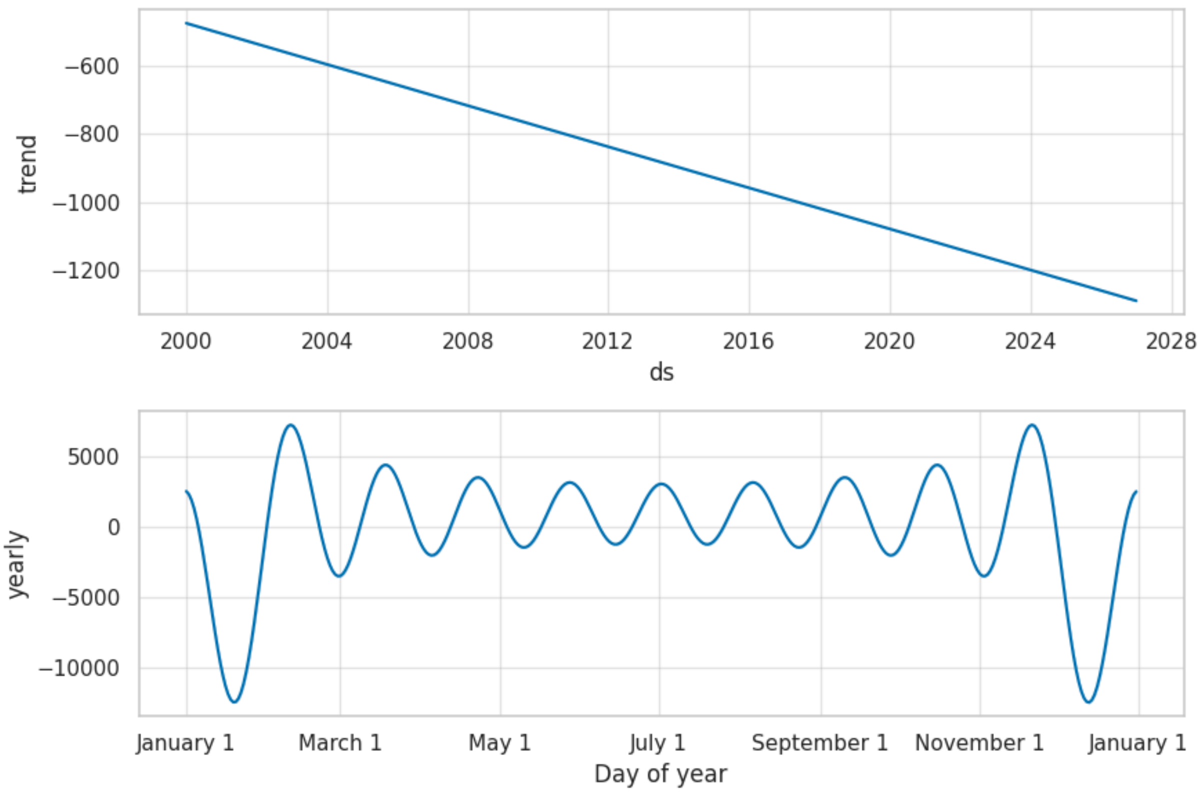
- In summary, the visualization indicates that the basic ARIMA model successfully identifies and projects the downward trend in aviation accidents. However, its ability to precisely predict the specific accident count for each year in the test set may be limited, as evidenced by the smoothing effect on fluctuations. The evaluation metrics (RMSE, MAE) provide a more objective measure of this performance.

In [94]:
```
# Import required libraries
import pandas as pd
import matplotlib.pyplot as plt
from prophet import Prophet
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Ensure event_date is datetime
df['event_date'] = pd.to_datetime(df['event_date'], errors='coerce')

# Filter data from 2000 to 2022
df_filtered = df[(df['event_date'].dt.year >= 2000) & (df['event_date'].dt.year <=

# Aggregate yearly accident counts
yearly_accidents = df_filtered.set_index('event_date').resample('Y').size()
yearly_accidents.index = yearly_accidents.index.year

# Prepare data for Prophet
prophet_df = yearly_accidents.reset_index()
prophet_df.columns = ['ds', 'y']
prophet_df['ds'] = pd.to_datetime(prophet_df['ds'], format='%Y')
```

```
# Fit Prophet model
prophet_model = Prophet(yearly_seasonality=True)
prophet_model.fit(prophet_df)

# Create future dataframe (next 5 years)
future = prophet_model.make_future_dataframe(periods=5, freq='Y')

# Forecast
forecast = prophet_model.predict(future)

# Plot forecast
fig = prophet_model.plot(forecast)
plt.title("Prophet Forecast - Aviation Accidents (2000-2027)")
plt.xlabel("Year")
plt.ylabel("Accident Count")
plt.tight_layout()
plt.show()

# Plot forecast components
prophet_model.plot_components(forecast)
plt.tight_layout()
plt.show()

# === In-sample Evaluation (2000-2022) ===

# Add year column to both Prophet forecast and original data
forecast['year'] = forecast['ds'].dt.year
prophet_df['year'] = prophet_df['ds'].dt.year

# Ensure alignment by merging on year
merged = pd.merge(prophet_df[['year', 'y']], forecast[['year', 'yhat']], on='year',

# Extract matched true and predicted values
y_true = merged['y'].values
y_pred = merged['yhat'].values

# Evaluation
mae = mean_absolute_error(y_true, y_pred)
rmse = np.sqrt(mean_squared_error(y_true, y_pred))

print("=== Prophet In-Sample Evaluation (2000-2022) ===")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
```

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True
to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to
override this.
INFO:prophet:n_changepoints greater than number of observations. Using 17.
DEBUG:cmdstanpy:input tempfile: /tmp/tmprzb1drvf/pu9hkd61.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmprzb1drvf/9e9suuxr.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan
_model/prophet_model.bin', 'random', 'seed=55755', 'data', 'file=/tmp/tmprzb1drvf/pu
9hkd61.json', 'init=/tmp/tmprzb1drvf/9e9suuxr.json', 'output', 'file=/tmp/tmprzb1drv
f/prophet_model9u3w6q_5/prophet_model-20250527081918.csv', 'method=optimize', 'algor
ithm=newton', 'iter=10000']
08:19:18 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
08:19:18 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```






```
=== Prophet In-Sample Evaluation (2000–2022) ===
Mean Absolute Error (MAE): 79.96
Root Mean Squared Error (RMSE): 100.41
```

Image 1: "Prophet Forecast - Aviation Accidents (2000-2027)"

Observations:

1. Declining Trend: The historical data (black dots and blue line) from 2000 to around 2023 shows a general downward trend in aviation accidents.

Seasonality/Fluctuations: While there's a downward trend, there are clear year-to-year fluctuations. For example, there's a noticeable dip around 2008 and a more significant dip around 2020.

2. Confidence Interval (Shaded Area): The blue shaded area represents the confidence interval (likely 95% confidence). It indicates the range within which the actual accident count is expected to fall. The interval widens as the forecast horizon extends, reflecting increased uncertainty.

Forecast (2024-2027): The blue line continues the downward trend into the forecast period. The forecast predicts a continued decrease in aviation accidents, though with more uncertainty as indicated by the widening confidence band. 2020 Dip: There's a very sharp and significant drop in accident count around 2020, likely attributable to the global impact of the COVID-19 pandemic on air travel. Post-2020 Rebound/Adjustment: After the 2020 dip, there's a slight rebound/increase in 2021 before the trend appears to continue downwards. Insights:

3. Effectiveness of Safety Measures: The overall downward trend suggests that aviation safety measures, technological advancements, and operational improvements over the years have been effective in reducing the number of accidents.

Impact of External Factors: The sharp dip in 2020 highlights how significant external events (like a global pandemic affecting travel) can drastically alter trends and present challenges for forecasting.

4. Uncertainty in Long-Term Forecasts: The widening confidence interval emphasizes that predictions further into the future (e.g., 2026-2027) are inherently less certain than short-term predictions.

5. Prophet Model Suitability: The use of a "Prophet Forecast" suggests the model is designed to handle trends and seasonality, which appear to be present in this data.

Base Rate Fallacy Consideration: While accidents are declining, it's important to consider if this is also reflective of a decline in total flight hours or if the accident rate per flight hour is decreasing. This chart only shows raw accident count.

Image 2: "ARIMA Forecast - Yearly Accident Count (2000-2022)

Observations:

1. Similar Historical Trend: The historical data (blue line with dots) from 2000 to 2021 shows a general decreasing trend in yearly accident count, similar to the Prophet forecast graph.

2. Pre-2020 Volatility: Before 2020, there are significant fluctuations, with peaks and troughs (e.g., 2001, 2006, 2011, 2016).

3. Sharp 2020 Drop: A very pronounced dip occurs in 2020, reaching the lowest point on the graph, likely due to the COVID-19 pandemic.

4. 2021 Recovery: There's a noticeable rebound in accident count in 2021 after the 2020 low.

5. ARIMA Forecast (2022-2027): The red dashed line represents the ARIMA forecast. It predicts a relatively stable number of accidents from 2022 to 2027, hovering just below 1600.

6. Lack of Confidence Interval: Unlike the Prophet forecast, this ARIMA forecast does not show a confidence interval (shaded area), making it harder to assess the uncertainty of the prediction visually.

7. No Actual Data in Forecast Period: There are no actual data points (dots) in the forecast period (2022-2027) to compare against the prediction.

Insights:

- Model Differences: The ARIMA model predicts a relatively flat trend for the forecast period (2022-2027), whereas the Prophet model from the other image predicted a continued decline. This difference highlights how different forecasting models can yield varying results based on their underlying assumptions and how they capture trends and seasonality.
- Recovery Stabilization: The ARIMA model seems to suggest that after the 2020 dip and 2021 rebound, the accident count might stabilize at a level higher than the Prophet model's later predictions but still lower than pre-pandemic levels.

Limitations Without Confidence Interval: The absence of a confidence interval makes it difficult to understand the range of possible outcomes and the inherent uncertainty of this ARIMA forecast.

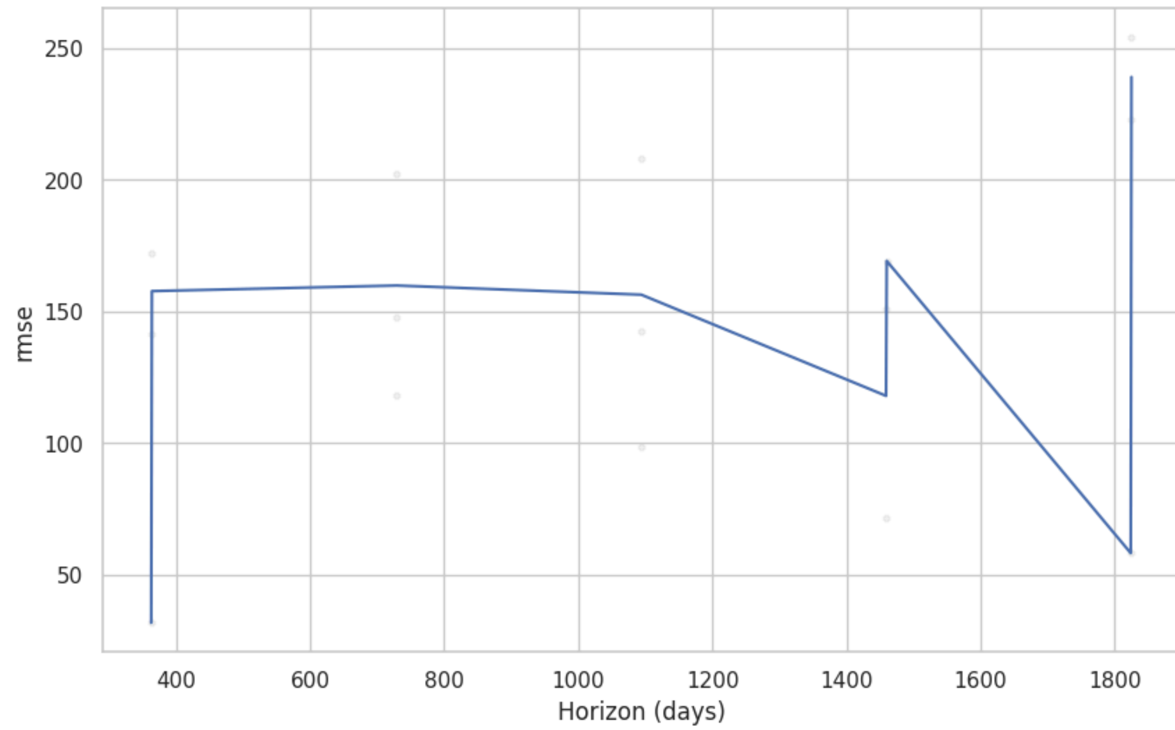
- Impact of 2020 Anomaly: Both models have to grapple with the significant and likely anomalous dip in 2020, which can heavily influence future predictions. How each model treats this outlier (e.g., as a temporary shock vs. a shift in trend) will affect its forecast.

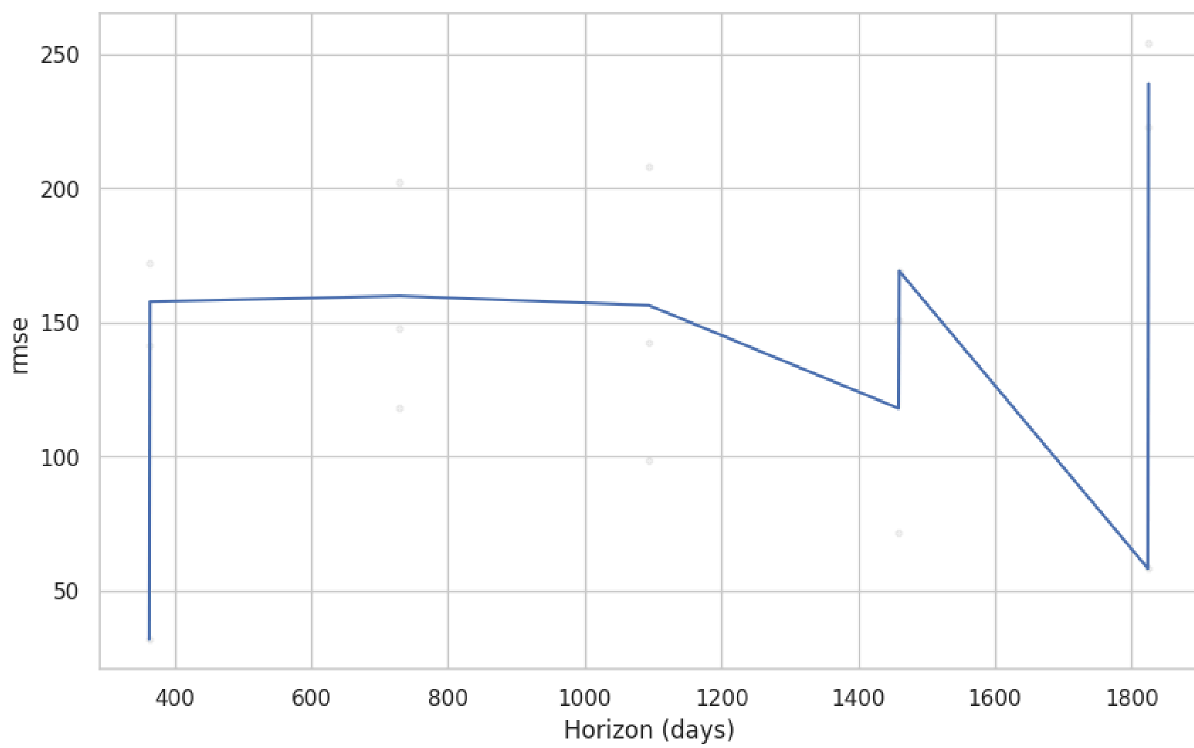
Overall Comparison and Accuracy Notes:

- Differing Predictions: The two forecasts, while both based on historical aviation accident data, offer different future trajectories. The Prophet model predicts a continued decline, while the ARIMA model predicts a stabilization.
- Accuracy Assessment: Without actual data for the forecast periods (especially post-2021/2022), it's impossible to definitively say which prediction is "accurate." The true accuracy would be determined by comparing the forecasted values with the actual accident counts in the coming years.
- Model Strengths: Prophet is often good with daily/weekly data and handling holidays/special events (which 2020 might be considered). ARIMA is a classic for time series, good for capturing autocorrelations and trends.
- Data Resolution: The images appear to be yearly data. Higher frequency data (e.g., monthly) could potentially lead to more nuanced forecasts.

In [96]:
```
from prophet.plot import plot_cross_validation_metric
plot_cross_validation_metric(df_cv, metric='rmse')
```

Out[96]:





Conclusion

This project provided a comprehensive analysis of aviation accidents using historical data from 1962 to 2022. Through exploratory data analysis (EDA), we identified key patterns and trends in accident occurrences, such as seasonal fluctuations, high-risk aircraft categories, and common phases of flight associated with severe injuries. The dataset was then enriched through feature engineering to improve model performance.

The forecasting component aimed to predict future accident trends. Time series models provided valuable insights into long-term patterns and potential future risks, which can help stakeholders in resource planning and preventive measures.

Overall, the integration of data preprocessing, and forecasting created a well-rounded pipeline that can support safety improvements in the aviation industry.

Recommendations

1. Enhance Data Collection:

- Improve data completeness for missing or underreported fields (e.g., exact weather conditions).
- Include human factors data (e.g., pilot experience, fatigue) if available, as these are critical for understanding accident causes.

2. Model Improvement:

- Use advanced time series models such as LSTM, or SARIMA for more accurate forecasting with seasonal components.

3. Feature Expansion:

- Incorporate additional geospatial data (e.g., flight routes, airport proximity) to analyze location-based risks.
- Add external factors such as aviation regulation changes or major events (e.g., pandemics, economic downturns).

4. Deployment & Monitoring:

- Develop a dashboard (e.g., in Tableau or Power BI) to visualize key trends and model predictions in real time.
- Set up periodic model retraining and evaluation as new data becomes available to maintain accuracy.

5. Industry Collaboration:

- Share insights with aviation authorities, maintenance crews, and training programs to tailor interventions based on model outputs.
- Encourage partnerships with government agencies for more granular data access and policy formulation.