In [1]:
```
import pandas as pd
df = pd.read_csv("aviation_project/Aviation_Data.csv")
df.head()
```

```
C:\Users\pc\miniconda3\envs\learn-env\lib\site-packages\IPython\core\interactiveshell.py:3145: DtypeWarning:
Columns (6,7,28) have mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[1]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitude	Longitude	Airport.Code	Airport.
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	United States	NaN	NaN	NaN	
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	United States	NaN	NaN	NaN	
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	United States	36.9222	-81.8781	NaN	
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	United States	NaN	NaN	NaN	
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	United States	NaN	NaN	NaN	

5 rows × 31 columns

In [2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90348 entries, 0 to 90347
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Event.Id                88889 non-null  object
 1   Investigation.Type      90348 non-null  object
 2   Accident.Number         88889 non-null  object
 3   Event.Date              88889 non-null  object
 4   Location                88837 non-null  object
 5   Country                 88663 non-null  object
 6   Latitude                34382 non-null  object
 7   Longitude               34373 non-null  object
 8   Airport.Code            50249 non-null  object
 9   Airport.Name            52790 non-null  object
 10  Injury.Severity         87889 non-null  object
 11  Aircraft.damage         85695 non-null  object
 12  Aircraft.Category       32287 non-null  object
 13  Registration.Number     87572 non-null  object
```

#Rows=90,348 & 31 Columns Columns monstly strings and float missing data = latitude, Aircraft.category & injury related fields.

Potential issues:

'.' in column names Non standard datatypes Many nulls in some columns eg Schedule, FAR.Description

In [3]:
```
df.shape
```

Out[3]:
```
(90348, 31)
```

In [4]:
```
df.describe(include='all')
```

Out[4]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitude	Longitude	Airport.Code	A
count	88889	90348	88889	88889	88837	88663	34382	34373	50249	
unique	87951	71	88863	14782	27758	219	25592	27156	10375	
top	20001214X45071	Accident	DCA23WA071	2000-07-08	ANCHORAGE, AK	United States	332739N	0112457W	NONE	
freq	3	85015	2	25	434	82248	19	24	1488	
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

11 rows × 31 columns

Step: Column Name Cleaning

To improve usability, we replaced all periods `.` in column names with underscores `_` using:

In [5]:
```
df.columns = df.columns.str.replace('.', '_', regex=False)
df.columns[:10]
```

Out[5]:
```
Index(['Event_Id', 'Investigation_Type', 'Accident_Number', 'Event_Date',
       'Location', 'Country', 'Latitude', 'Longitude', 'Airport_Code',
       'Airport_Name'],
      dtype='object')
```

Columns with missing values

In [6]:
```
df.isnull().sum().sort_values(ascending=False).head(10)
```

Out[6]:
```
Schedule              77766
Air_carrier           73700
FAR_Description       58325
Aircraft_Category     58061
Longitude             55975
Latitude              55966
Airport_Code          40099
Airport_Name          37558
Broad_phase_of_flight 28624
Publication_Date      16689
dtype: int64
```

drop schedule Column

In [7]:
```
df.drop(columns='Schedule', inplace=True)
```

In [8]:
```
df.isnull().sum().sort_values(ascending=False).head(10)
```

Out[8]:
```
Air_carrier              73700
FAR_Description          58325
Aircraft_Category        58061
Longitude                55975
Latitude                 55966
Airport_Code             40099
Airport_Name             37558
Broad_phase_of_flight    28624
Publication_Date         16689
Total_Serious_Injuries   13969
dtype: int64
```

Explore Columns:

- Air_carrier
- FAR_Description
- Aircraft_Category

```
### Aircraft_Category
```

In [9]:
```
df['Aircraft_Category'].value_counts(dropna=False)
```

Out[9]:
```
NaN                  58061
Airplane             27617
Helicopter            3440
Glider                 508
Balloon                231
Gyrocraft              173
Weight-Shift           161
Powered Parachute       91
Ultralight              30
Unknown                 14
WSFT                     9
Powered-Lift             5
Blimp                    4
UNK                      2
ULTR                     1
Rocket                   1
Name: Aircraft_Category, dtype: int64
```

```
###Air_carrier
```

In [10]:
```
df['Air_carrier'].value_counts(dropna=False).head(10)
```

Out[10]:
```
NaN                      73700
Pilot                      258
American Airlines           90
United Airlines             89
Delta Air Lines             53
SOUTHWEST AIRLINES CO       42
DELTA AIR LINES INC         37
AMERICAN AIRLINES INC       29
ON FILE                     27
Continental Airlines        27
Name: Air_carrier, dtype: int64
```

```
###FAR_Description -common type of regulations under whhich the aircraft was opersting.
```

In [11]:
```
df['FAR_Description'].value_counts(dropna=False)
```

Out[11]:
```
NaN                              58325
091                              18221
Part 91: General Aviation         6486
NUSN                              1584
NUSC                              1013
137                               1010
135                                746
121                                679
Part 137: Agricultural             437
UNK                                371
Part 135: Air Taxi & Commuter      298
PUBU                               253
129                                246
Part 121: Air Carrier              165
133                                107
Part 129: Foreign                  100
Non-U.S., Non-Commercial            97
Non-U.S., Commercial                93
Part 133: Rotorcraft Ext. Load      32
Unknown                             22
Public Use                          19
091K                                14
ARMF                                 8
125                                  5
Part 125: 20+ Pax,6000+ lbs          5
107                                  4
Public Aircraft                      2
103                                  2
Part 91F: Special Flt Ops.           1
Part 91 Subpart K: Fractional        1
Armed Forces                         1
437                                  1
Name: FAR_Description, dtype: int64
```

```
###Aircraft_Category
-Most common: Airplane, then Helicopter
-Rare types: Glider, Blimp, Rocket, etc.
-Over 58k missing values (NaN)

Air_carrier
-Most entries are "Pilot" (likely personal or small charter aircraft)
-Second by commercial carriers like American Airlines, United, Delta
-Some duplicate entries: "DELTA AIR LINES INC" vs "Delta Air Lines" (need to clean)

FAR_Description
-Part 91: General Aviation — dominates
-Over 58,000 missing values — we'll either drop them or impute based on known patterns
```

Filtering

In [12]:
```
# Maintain rows with at least one type of injury recorded
df_injured = df[
    (df['Total_Fatal_Injuries'].fillna(0) > 0) |
    (df['Total_Serious_Injuries'].fillna(0) > 0) |
    (df['Total_Minor_Injuries'].fillna(0) > 0)
]
df_injured.shape
```

Out[12]: (40491, 30)

Grouping Aircraft category and total fatalities incurred.

In [13]:
```
# Group by
fatal_by_aircraft = df_injured.groupby('Aircraft_Category')['Total_Fatal_Injuries'].sum().sort_values(ascendi
fatal_by_aircraft
```

Out[13]:
```
Aircraft_Category
Airplane             16029.0
Helicopter            1778.0
Glider                  99.0
Weight-Shift            67.0
Gyrocraft               44.0
Balloon                 43.0
Unknown                 16.0
Powered Parachute       15.0
WSFT                    10.0
Ultralight              10.0
Rocket                   1.0
ULTR                     0.0
Powered-Lift             0.0
Blimp                    0.0
Name: Total_Fatal_Injuries, dtype: float64
```

BUSINESS INTERPRETATION.

Airplanes account for the highest number of fatalities, but also represent the largest proportion of aircraft in operation. Helicopters are second in both frequency and fatality count. Small categories like Gliders, Balloons, and Powered Parachutes show lower fatalities, but may be less scalable for commercial use.

###Frequency calculation.

In [14]:
```
# Count frquency of aircraft type
aircraft_counts = df['Aircraft_Category'].value_counts()
aircraft_counts
```

Out[14]:
```
Airplane             27617
Helicopter            3440
Glider                 508
Balloon                231
Gyrocraft              173
Weight-Shift           161
Powered Parachute       91
Ultralight              30
Unknown                 14
WSFT                     9
Powered-Lift             5
Blimp                    4
UNK                      2
ULTR                     1
Rocket                   1
Name: Aircraft_Category, dtype: int64
```

In [15]:
```
# Fatalities per aircraft type
fatal_by_aircraft = df_injured.groupby('Aircraft_Category')['Total_Fatal_Injuries'].sum()
fatal_by_aircraft
```

Out[15]:
```
Aircraft_Category
Airplane             16029.0
Balloon                 43.0
Blimp                    0.0
Glider                  99.0
Gyrocraft               44.0
Helicopter            1778.0
Powered Parachute       15.0
Powered-Lift             0.0
Rocket                   1.0
ULTR                     0.0
Ultralight              10.0
Unknown                 16.0
WSFT                    10.0
Weight-Shift            67.0
Name: Total_Fatal_Injuries, dtype: float64
```

In [16]:
```
# Combine into one DataFrame
risk_df = pd.DataFrame({
    'Count': aircraft_counts,
    'Fatalities': fatal_by_aircraft
})
risk_df
```

Out[16]:

	Count	Fatalities
Airplane	27617	16029.0
Balloon	231	43.0
Blimp	4	0.0
Glider	508	99.0
Gyrocraft	173	44.0
Helicopter	3440	1778.0
Powered Parachute	91	15.0
Powered-Lift	5	0.0
Rocket	1	1.0
ULTR	1	0.0
UNK	2	NaN
Ultralight	30	10.0
Unknown	14	16.0
WSFT	9	10.0
Weight-Shift	161	67.0

In [17]:
```
# Fill NaN fatalities with 0
risk_df['Fatalities'] = risk_df['Fatalities'].fillna(0)
risk_df
```

Out[17]:

	Count	Fatalities
Airplane	27617	16029.0
Balloon	231	43.0
Blimp	4	0.0
Glider	508	99.0
Gyrocraft	173	44.0
Helicopter	3440	1778.0
Powered Parachute	91	15.0
Powered-Lift	5	0.0
Rocket	1	1.0
ULTR	1	0.0
UNK	2	0.0
Ultralight	30	10.0
Unknown	14	16.0
WSFT	9	10.0
Weight-Shift	161	67.0

In [18]:

```
#  fatality rate per 1000 flights
risk_df['Fatality_Rate'] = (risk_df['Fatalities'] / risk_df['Count']) * 1000
risk_df['Fatality_Rate']
```

Out[18]:

```
Airplane              580.403375
Balloon               186.147186
Blimp                   0.000000
Glider                194.881890
Gyrocraft             254.335260
Helicopter            516.860465
Powered Parachute     164.835165
Powered-Lift            0.000000
Rocket               1000.000000
ULTR                    0.000000
UNK                     0.000000
Ultralight            333.333333
Unknown              1142.857143
WSFT                 1111.111111
Weight-Shift          416.149068
Name: Fatality_Rate, dtype: float64
```

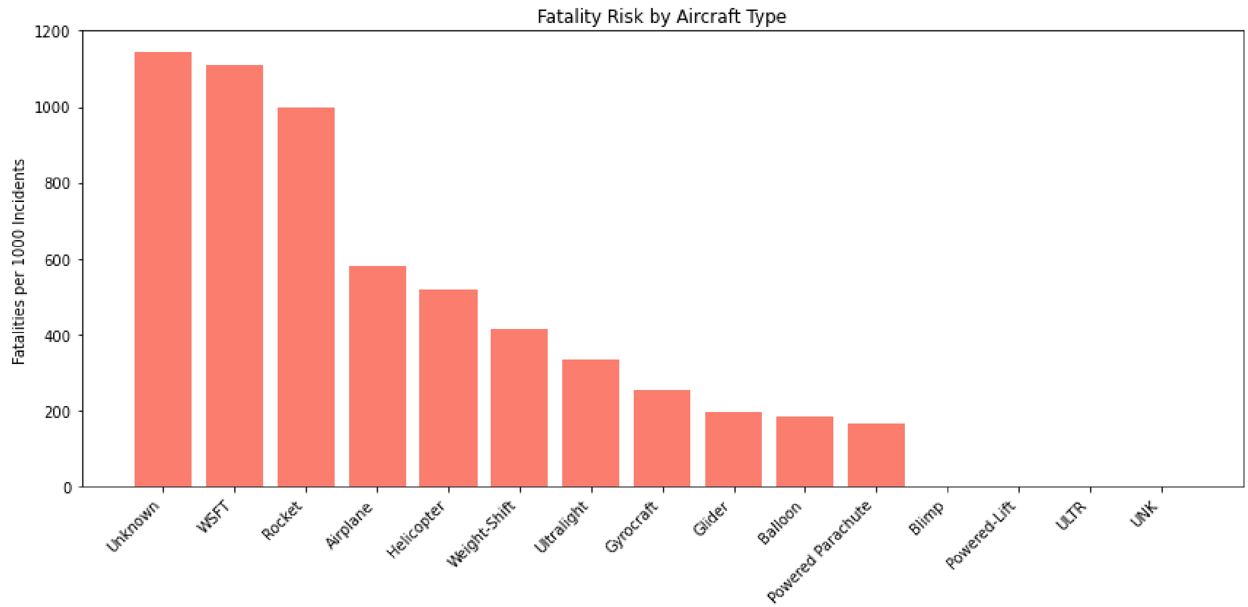
In [19]:

```
# Sort by fatality rate
risk_df.sort_values('Fatality_Rate', ascending=False)
```

Out[19]:

	Count	Fatalities	Fatality_Rate
Unknown	14	16.0	1142.857143
WSFT	9	10.0	1111.111111
Rocket	1	1.0	1000.000000
Airplane	27617	16029.0	580.403375
Helicopter	3440	1778.0	516.860465
Weight-Shift	161	67.0	416.149068
Ultralight	30	10.0	333.333333
Gyrocraft	173	44.0	254.335260
Glider	508	99.0	194.881890
Balloon	231	43.0	186.147186
Powered Parachute	91	15.0	164.835165
Blimp	4	0.0	0.000000
Powered-Lift	5	0.0	0.000000
ULTR	1	0.0	0.000000
UNK	2	0.0	0.000000

In [20]:
```
import matplotlib.pyplot as plt
risk_df_sorted = risk_df.sort_values('Fatality_Rate', ascending=False)
plt.figure(figsize=(12, 6))
plt.bar(risk_df_sorted.index, risk_df_sorted['Fatality_Rate'], color='salmon')
plt.xticks(rotation=45, ha='right')
plt.ylabel('Fatalities per 1000 Incidents')
plt.title('Fatality Risk by Aircraft Type')
plt.tight_layout()
plt.show()
```



In [21]:
```
df.columns.tolist()
```

Out[21]:
```
['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',
 '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']
```

In [22]:
```
columns_to_drop = ['Air_carrier', 'FAR_Description', 'Airport_Name']  # 'Schedule' removed
df_cleaned = df.drop(columns=columns_to_drop)
```

In [23]:
```
df_cleaned.shape
df_cleaned.columns.tolist()
```

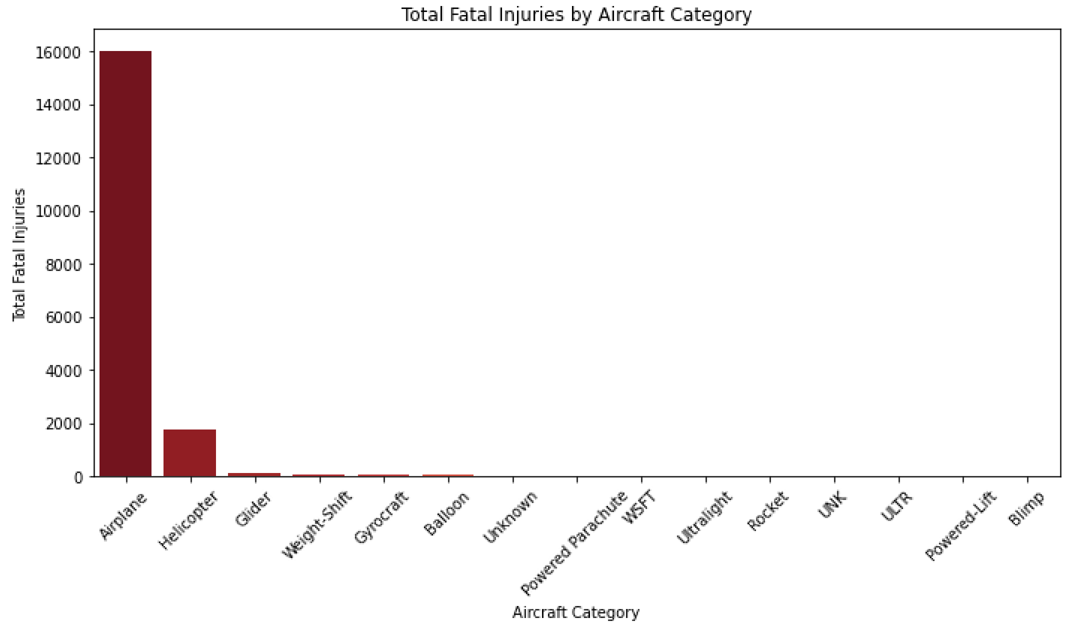
Out[23]:
```
['Event_Id',
 'Investigation_Type',
 'Accident_Number',
 'Event_Date',
 'Location',
 'Country',
 'Latitude',
 'Longitude',
 'Airport_Code',
 'Injury_Severity',
 'Aircraft_damage',
 'Aircraft_Category',
 'Registration_Number',
 'Make',
 'Model',
 'Amateur_Built',
 'Number_of_Engines',
 'Engine_Type',
 'Purpose_of_flight',
 'Total_Fatal_Injuries',
 'Total_Serious_Injuries',
 'Total_Minor_Injuries',
 'Total_Uninjured',
 'Weather_Condition',
 'Broad_phase_of_flight',
 'Report_Status',
 'Publication_Date']
```

In [24]:

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

# Group and sort
fatal_by_category = df_cleaned.groupby('Aircraft_Category')['Total_Fatal_Injuries'].sum().sort_values(ascendi

# Plot
plt.figure(figsize=(10,6))
sns.barplot(x=fatal_by_category.index, y=fatal_by_category.values, palette='Reds_r')
plt.xticks(rotation=45)
plt.title('Total Fatal Injuries by Aircraft Category')
plt.ylabel('Total Fatal Injuries')
plt.xlabel('Aircraft Category')
plt.tight_layout()
plt.show()
```



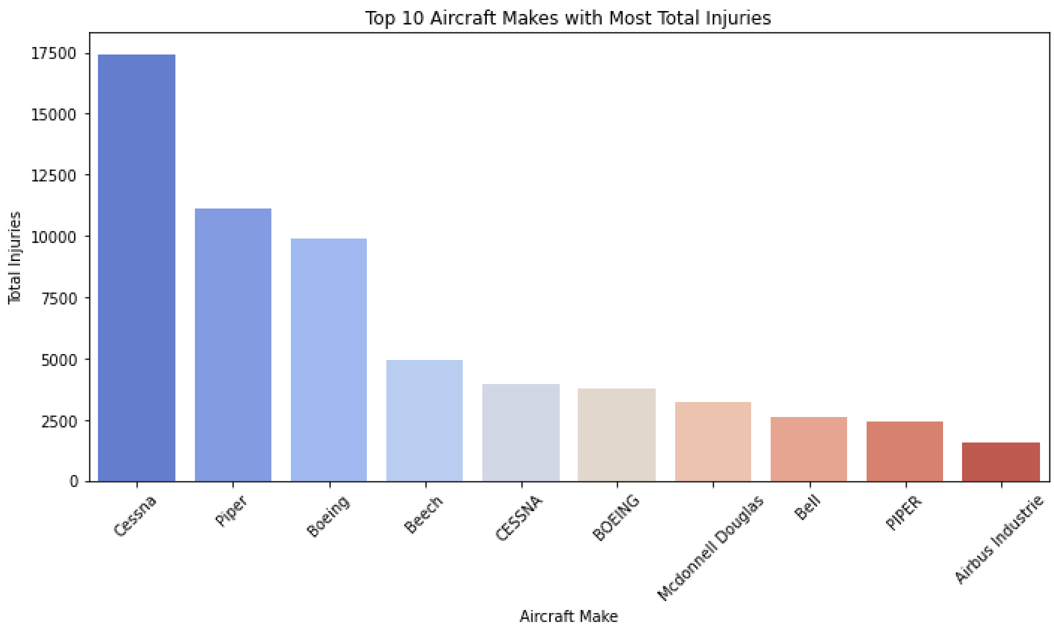
EDA

- some aircraft categories like "Powered-Lift", "Blimp", or "Rocket" don’t even have injury data because:
    - They're rarely used, especially in commercial settings.
    - Accidents involving them are extremely uncommon or underreported.
    - Some might be military-only or experimental, hence not logged in public dataset

In [25]:

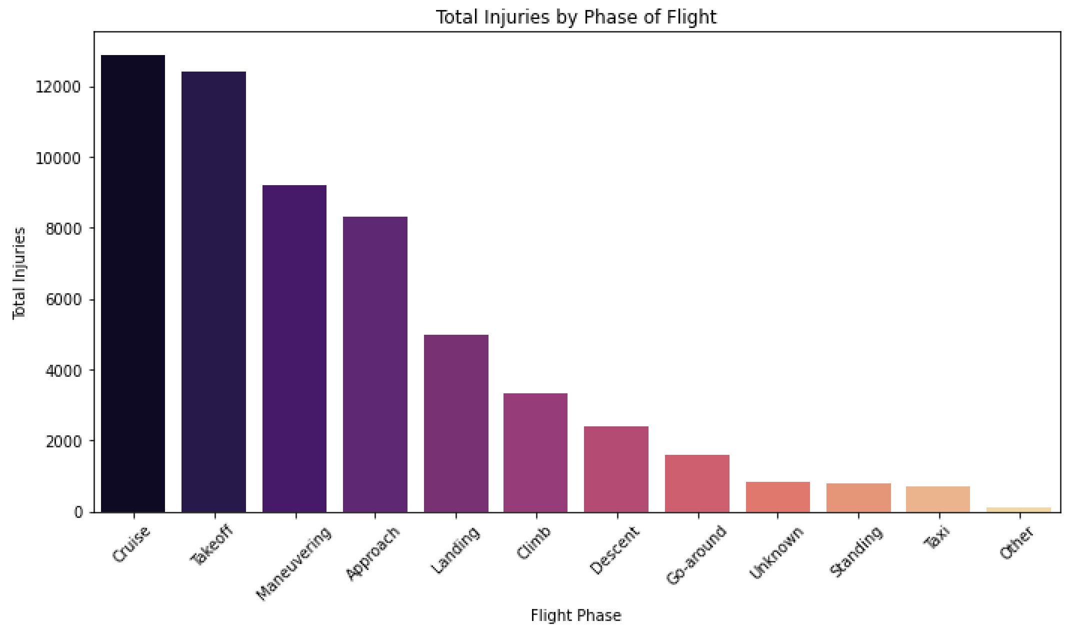
```
# Calculate total injuries
df_cleaned['Total_Injuries'] = df_cleaned[['Total_Fatal_Injuries', 'Total_Serious_Injuries', 'Total_Minor_Inj
injuries_by_make = df_cleaned.groupby('Make')['Total_Injuries'].sum().sort_values(ascending=False).head(10)

plt.figure(figsize=(10,6))
sns.barplot(x=injuries_by_make.index, y=injuries_by_make.values, palette='coolwarm')
plt.xticks(rotation=45)
plt.title('Top 10 Aircraft Makes with Most Total Injuries')
plt.ylabel('Total Injuries')
plt.xlabel('Aircraft Make')
plt.tight_layout()
plt.show()
```



In [26]:

```
# Group by flight phase and sum total injuries
phase_risks = df_cleaned.groupby('Broad_phase_of_flight')['Total_Injuries'].sum().sort_values(ascending=False
plt.figure(figsize=(10,6))
sns.barplot(x=phase_risks.index, y=phase_risks.values, palette='magma')
plt.xticks(rotation=45)
plt.title('Total Injuries by Phase of Flight')
plt.ylabel('Total Injuries')
plt.xlabel('Flight Phase')
plt.tight_layout()
plt.show()
```



In [27]:

```
# mean, median, std of Total_Fatal_Injuries
risk_stats = df_cleaned.groupby('Aircraft_Category')['Total_Fatal_Injuries'].agg(['mean', 'median', 'std']).s
risk_stats
```

Out[27]:

Aircraft_Category	mean	median	std
Unknown	1.142857	0.5	1.994498
WSFT	1.111111	1.0	0.927961
Rocket	1.000000	1.0	NaN
Airplane	0.655529	0.0	5.943002
Helicopter	0.582569	0.0	1.383069
Ultralight	0.416667	0.0	0.653863
Weight-Shift	0.416149	0.0	0.637956
Gyrocraft	0.287582	0.0	0.546197
Glider	0.235154	0.0	0.501711
Balloon	0.223958	0.0	1.293094
Powered Parachute	0.164835	0.0	0.428531
Blimp	0.000000	0.0	NaN
Powered-Lift	0.000000	0.0	0.000000
ULTR	0.000000	0.0	NaN
UNK	0.000000	0.0	0.000000

In [28]:

```
# Drop rows with NaN in statistics
risk_stats_cleaned = risk_stats.dropna()
risk_stats_cleaned
```

Out[28]:

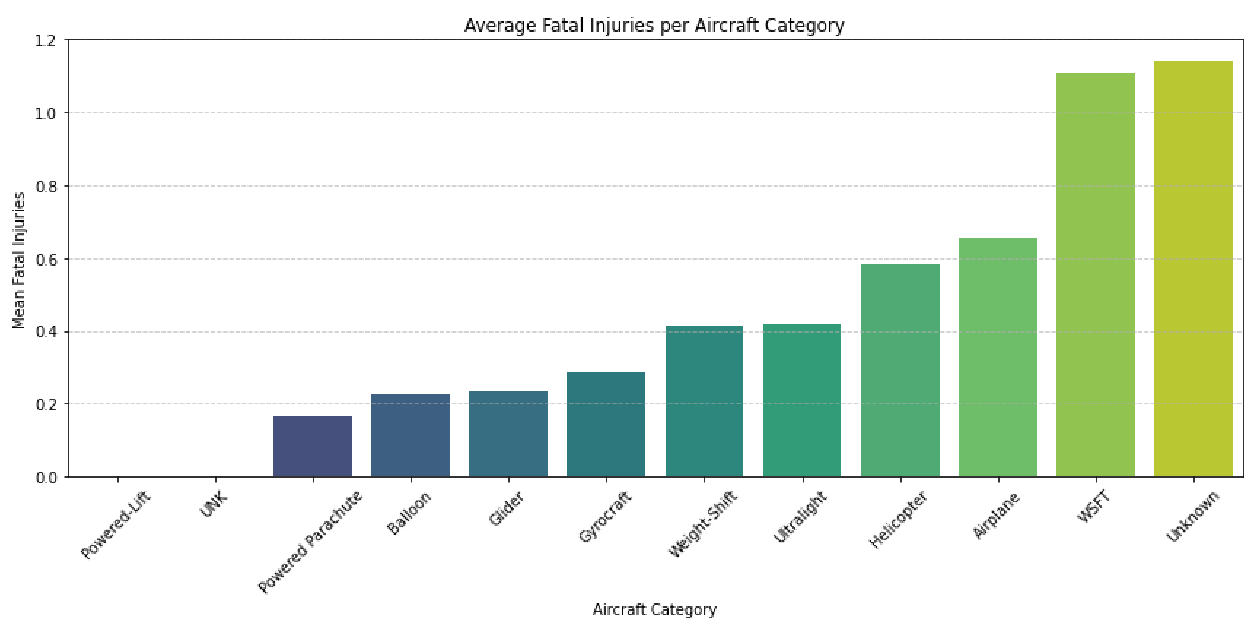
Aircraft_Category	mean	median	std
Unknown	1.142857	0.5	1.994498
WSFT	1.111111	1.0	0.927961
Airplane	0.655529	0.0	5.943002
Helicopter	0.582569	0.0	1.383069
Ultralight	0.416667	0.0	0.653863
Weight-Shift	0.416149	0.0	0.637956
Gyrocraft	0.287582	0.0	0.546197
Glider	0.235154	0.0	0.501711
Balloon	0.223958	0.0	1.293094
Powered Parachute	0.164835	0.0	0.428531
Powered-Lift	0.000000	0.0	0.000000
UNK	0.000000	0.0	0.000000

In [29]:

```
import matplotlib.pyplot as plt
import seaborn as sns
risk_stats_cleaned_sorted = risk_stats_cleaned.sort_values(by='mean')
plt.figure(figsize=(12, 6))
sns.barplot(x=risk_stats_cleaned_sorted.index, y=risk_stats_cleaned_sorted['mean'], palette='viridis')

plt.title('Average Fatal Injuries per Aircraft Category')
plt.ylabel('Mean Fatal Injuries')
plt.xlabel('Aircraft Category')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

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



1. Low-Risk Aircraft (Good for Business)
-lowest bars on the graph (i.e., aircraft types with the lowest mean fatal injuries).

Powered Parachute, Weight-Shift, Ultralight, Glider

-These are candidates for low operational risk and good entry points for the company.

2. High-Risk Aircraft
The highest means and/or very high standard deviation, e.g.:
-Airplane & Helicopter

-These might require more certification, experienced pilots, and high insurance premiums

Business Recommendation

-Based on analysis of aircraft incident data, the categories such as Powered Parachute, Weight-Shift, and Glider consistently show the lowest mean fatal injuries, making them ideal for low-risk entry into the aviation industry.

-In contrast, Airplanes and Helicopters, while common, have significantly higher injury rates, suggesting the need for more stringent safety measures and risk assessments before investment

In [30]:

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

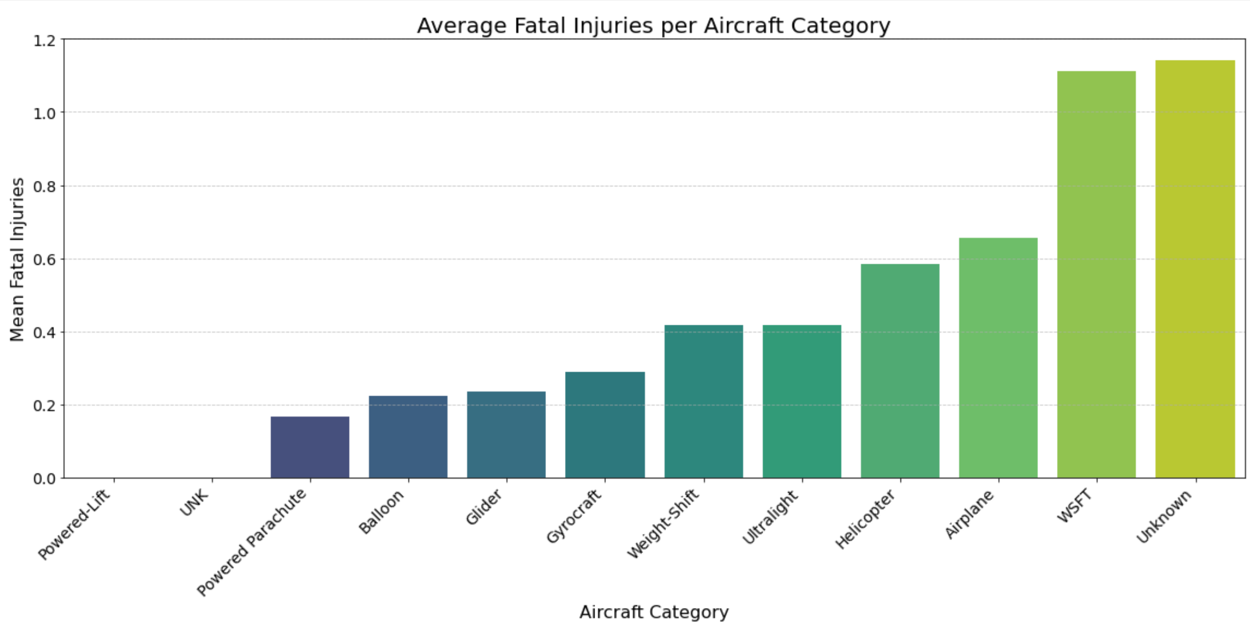
Injury serverity Distribution.

In [31]:

```
plt.figure(figsize=(16, 8))
sns.barplot(x=risk_stats_cleaned_sorted.index, y=risk_stats_cleaned_sorted['mean'], palette='viridis')

plt.title('Average Fatal Injuries per Aircraft Category', fontsize=20)
plt.xlabel('Aircraft Category', fontsize=16)
plt.ylabel('Mean Fatal Injuries', fontsize=16)
plt.xticks(rotation=45, fontsize=14, ha='right')
plt.yticks(fontsize=14)

plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

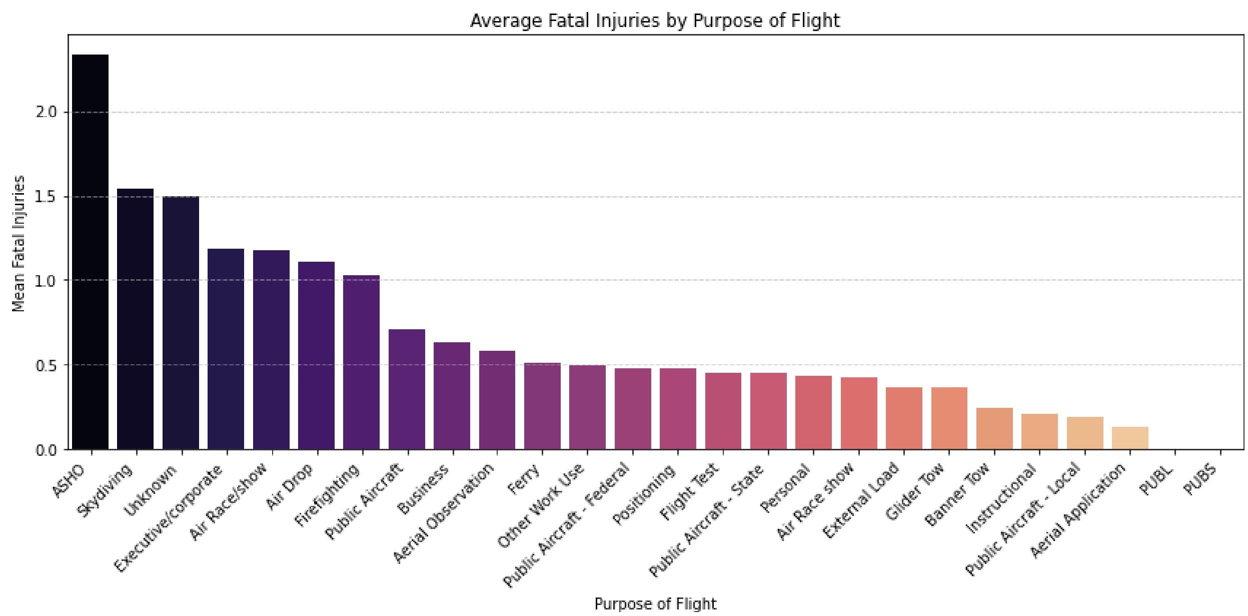


Injuries by Purpose of Flight

In [32]:

```
# Group and mean fatal injuries by Purpose_of_flight
purpose_injuries = df_cleaned.groupby('Purpose_of_flight')['Total_Fatal_Injuries'].mean().sort_values(ascendi

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(x=purpose_injuries.index, y=purpose_injuries.values, palette='magma')
plt.title('Average Fatal Injuries by Purpose of Flight')
plt.ylabel('Mean Fatal Injuries')
plt.xlabel('Purpose of Flight')
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.tight_layout()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

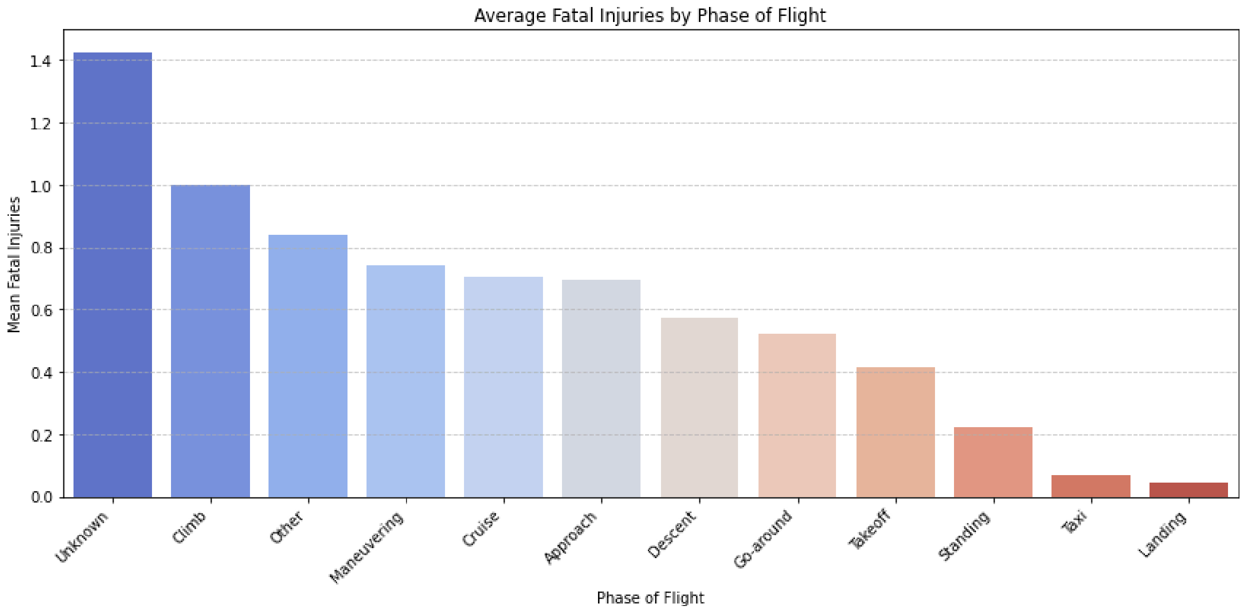


###Phase of flight Analysis

In [33]:

```
#groupby
phase_injuries = df_cleaned.groupby('Broad_phase_of_flight')['Total_Fatal_Injuries'].mean().sort_values(ascen

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(x=phase_injuries.index, y=phase_injuries.values, palette='coolwarm')
plt.title('Average Fatal Injuries by Phase of Flight')
plt.ylabel('Mean Fatal Injuries')
plt.xlabel('Phase of Flight')
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.tight_layout()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

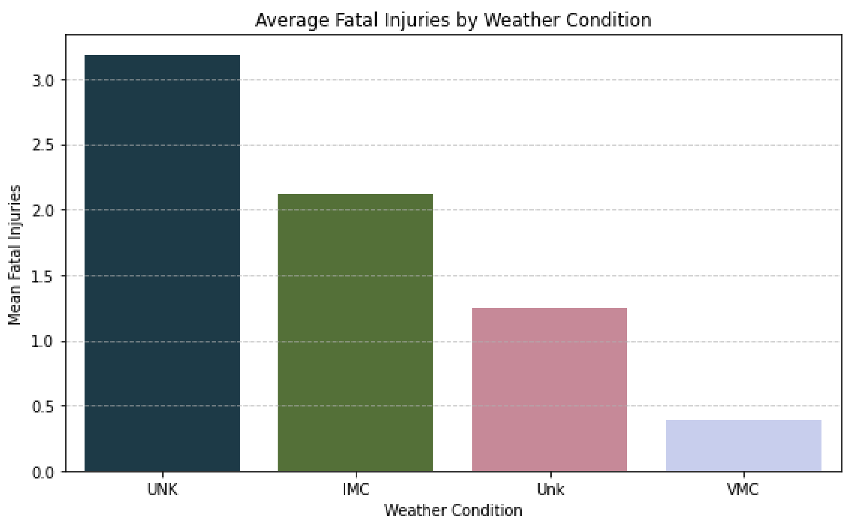


Weather Condition Analysis

In [34]:

```
weather_injuries = df_cleaned.groupby('Weather_Condition')['Total_Fatal_Injuries'].mean().sort_values(ascendi

# Plot
plt.figure(figsize=(8, 5))
sns.barplot(x=weather_injuries.index, y=weather_injuries.values, palette='cubehelix')
plt.title('Average Fatal Injuries by Weather Condition')
plt.ylabel('Mean Fatal Injuries')
plt.xlabel('Weather Condition')
plt.tight_layout()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



In [35]:

```
# Export purpose_of_flight summary
purpose_injuries.to_csv('purpose_fatal_injuries_summary.csv')
```

In [36]:

```
# Export phase of flight summary
phase_injuries.to_csv('phase_fatal_injuries_summary.csv')
```

In [37]:

```
# Export weather condition summary
weather_injuries.to_csv('weather_fatal_injuries_summary.csv')
```