In [310]:
```
# importing libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
sns.set(style="whitegrid")
```

In [311]:
```
# loading the dataset
df = pd.read_csv('../zippedData/AviationData.csv', encoding='latin1',low_me
```

In [312]:
```
# checking the shape
df.shape
```

Out[312]: (88889, 31)

In [313]:
```
# checking the head
df.head()
```

Out[313]:

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

5 rows × 31 columns

In [314]:
```
# checking the tail
df.tail()
```

Out[314]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	L
88884	20221227106491	Accident	ERA23LA093	2022-12-26	Annapolis, MD	United States	
88885	20221227106494	Accident	ERA23LA095	2022-12-26	Hampton, NH	United States	
88886	20221227106497	Accident	WPR23LA075	2022-12-26	Payson, AZ	United States	3
88887	20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, UT	United States	
88888	20221230106513	Accident	ERA23LA097	2022-12-29	Athens, GA	United States	

5 rows × 31 columns

In [315]:
```
# checking the columns
df.columns
```

Out[315]:
```
Index(['Event.Id', 'Investigation.Type', 'Accident.Number', 'Event.Date',
       'Location', 'Country', 'Latitude', 'Longitude', 'Airport.Code',
       'Airport.Name', 'Injury.Severity', 'Aircraft.damage',
       'Aircraft.Category', 'Registration.Number', 'Make', 'Model',
       'Amateur.Built', 'Number.of.Engines', 'Engine.Type', 'FAR.Descriptio
n',
       'Schedule', 'Purpose.of.flight', 'Air.carrier', 'Total.Fatal.Injurie
s',
       'Total.Serious.Injuries', 'Total.Minor.Injuries', 'Total.Uninjured',
       'Weather.Condition', 'Broad.phase.of.flight', 'Report.Status',
       'Publication.Date'],
      dtype='object')
```

In [316]:
```
df.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            50249 non-null  object 
 9   Airport.Name            52790 non-null  object 
 10  Injury.Severity         87889 non-null  object 
 11  Aircraft.damage         85695 non-null  object 
 12  Aircraft.Category       32287 non-null  object 
 13  Registration.Number     87572 non-null  object 
 14  Make                    88826 non-null  object 
 15  Model                   88797 non-null  object 
 16  Amateur.Built           88787 non-null  object 
 17  Number.of.Engines       82805 non-null  float64
 18  Engine.Type             81812 non-null  object 
 19  FAR.Description         32023 non-null  object 
 20  Schedule                12582 non-null  object 
 21  Purpose.of.flight       82697 non-null  object 
 22  Air.carrier             16648 non-null  object 
 23  Total.Fatal.Injuries    77488 non-null  float64
 24  Total.Serious.Injuries  76379 non-null  float64
 25  Total.Minor.Injuries    76956 non-null  float64
 26  Total.Uninjured         82977 non-null  float64
 27  Weather.Condition       84397 non-null  object 
 28  Broad.phase.of.flight   61724 non-null  object 
 29  Report.Status           82508 non-null  object 
 30  Publication.Date        75118 non-null  object 
dtypes: float64(5), object(26)
memory usage: 21.0+ MB
```

In [317]:
```
# statistical summary for all
df.describe(include='all')
```

Out[317]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Count
count	88889	88889	88889	88889	88837	886
unique	87951	2	88863	14782	27758	2
top	20001214X45071	Accident	DCA22WA089	1982-05-16	ANCHORAGE, AK	Unit Stat
freq	3	85015	2	25	434	822
mean	NaN	NaN	NaN	NaN	NaN	Na
std	NaN	NaN	NaN	NaN	NaN	Na
min	NaN	NaN	NaN	NaN	NaN	Na
25%	NaN	NaN	NaN	NaN	NaN	Na
50%	NaN	NaN	NaN	NaN	NaN	Na
75%	NaN	NaN	NaN	NaN	NaN	Na
max	NaN	NaN	NaN	NaN	NaN	Na

11 rows × 31 columns

Data Cleaning

In [318]:
```
# checking for the null values, from the highest to the lowest %
(df.isna().mean()*100).sort_values(ascending = False)
```

Out[318]:
```
Schedule                 85.845268
Air.carrier              81.271023
FAR.Description          63.974170
Aircraft.Category        63.677170
Longitude                61.330423
Latitude                 61.320298
Airport.Code             43.469946
Airport.Name             40.611324
Broad.phase.of.flight    30.560587
Publication.Date         15.492356
Total.Serious.Injuries   14.073732
Total.Minor.Injuries     13.424608
Total.Fatal.Injuries     12.826109
Engine.Type               7.961615
Report.Status             7.178616
Purpose.of.flight         6.965991
Number.of.Engines         6.844491
Total.Uninjured           6.650992
Weather.Condition         5.053494
Aircraft.damage           3.593246
Registration.Number       1.481623
Injury.Severity           1.124999
Country                   0.254250
Amateur.Built             0.114750
Model                     0.103500
Make                      0.070875
Location                  0.058500
Event.Date                0.000000
Accident.Number           0.000000
Investigation.Type        0.000000
Event.Id                  0.000000
dtype: float64
```

In [319]:
```
# dropping columns where missing_values >= 30%

cols_to_drop = ['Schedule', 'Air.carrier','FAR.Description','Aircraft.Categ
                'Longitude','Latitude','Airport.Code','Airport.Name','Broad.

df.drop(columns=cols_to_drop, inplace=True)
```

In [320]:
```
# checking the remaining columns
df.columns
```

Out[320]:
```
Index(['Event.Id', 'Investigation.Type', 'Accident.Number', 'Event.Date',
       'Location', 'Country', 'Injury.Severity', 'Aircraft.damage',
       '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',
       'Report.Status', 'Publication.Date'],
      dtype='object')
```

In [321]:
```
# drop other columns which are not important for the analysis
other_cols_to_drop = ['Accident.Number','Registration.Number','Amateur.Buil

df.drop(columns=other_cols_to_drop, inplace=True)

df.columns #columns to work with
```

Out[321]:
```
Index(['Event.Id', 'Investigation.Type', 'Event.Date', 'Location', 'Countr
y',
       'Injury.Severity', 'Aircraft.damage', 'Make', 'Model',
       'Number.of.Engines', 'Engine.Type', 'Purpose.of.flight',
       'Total.Fatal.Injuries', 'Total.Serious.Injuries',
       'Total.Minor.Injuries', 'Total.Uninjured', 'Weather.Condition'],
      dtype='object')
```

In [322]:
```
# renaming the columns for ease of analysis using df.rename()

renamed_col = {'Event.Id':'ID','Investigation.Type':'Type','Event.Date': 'D
               'Number.of.Engines':'Engines', 'Engine.Type':'Engine_Type',
              'Total.Serious.Injuries':'Serious_Injuries', 'Total.Minor.Inj
df.rename(columns=renamed_col, inplace= True,)

df.head(2)
```

Out[322]:

	ID	Type	Date	Location	Country	Injury_Severity	Damage_type	M
0	20001218X45444	Accident	1948-10-24	MOOSE CREEK, ID	United States	Fatal(2)	Destroyed	Stin
1	20001218X45447	Accident	1962-07-19	BRIDGEPORT, CA	United States	Fatal(4)	Destroyed	P

In [323]:
```
# checking for unique variables in the selected columnns for analysis
for colm in df:
    colm_val = df[colm].unique()
    print(f"{colm},'\n',{colm_val}","\n")
```

```
ID,'
',['20001218X45444' '20001218X45447' '20061025X01555' ... '2022122710649
7'
 '20221227106498' '20221230106513']

Type,'
',['Accident' 'Incident']

Date,'
',['1948-10-24' '1962-07-19' '1974-08-30' ... '2022-12-22' '2022-12-26'
 '2022-12-29']

Location,'
',['MOOSE CREEK, ID' 'BRIDGEPORT, CA' 'Saltville, VA' ... 'San Manual, A
Z'
 'Auburn Hills, MI' 'Brasnorte, ']

Country,'
',['United States' nan 'GULF OF MEXICO' 'Puerto Rico' 'ATLANTIC OCEAN'
```

In [324]:
```
# DATE
# changing the date to an appropriate format
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

# creating a seasons column as the accidents rate will vary depending on se

df['Month'] = df['Date'].dt.month
seasons = {1: 'Winter', 2: 'Winter',3: 'Spring', 4: 'Spring', 5: 'Spring',
           6: 'Summer', 7: 'Summer', 8: 'Summer', 9: 'Fall', 10: 'Fall',
           11: 'Fall', 12 : 'Winter'}
df['Season'] = df['Month'].map(seasons)

# since year-on-year analysis could be important, I'll create a new
# column out of Date column
df['Year'] = df['Date'].dt.year
```

In [325]:
```
# LOCATION
df['Location'].value_counts()
```

Out[325]:
```
ANCHORAGE, AK        434
MIAMI, FL            200
ALBUQUERQUE, NM      196
HOUSTON, TX          193
CHICAGO, IL          184
                    ...
S.E. OF MALTA, ID      1
ELBERON, VA            1
NOMAN'S LAND, MA       1
NORCO, LA              1
Bloemfontein, OF       1
Name: Location, Length: 27758, dtype: int64
```

In [326]:
```
# Splitting location in city and state
df['City'] = df['Location'].str.split(',').str[0]
df['State'] = df['Location'].str.split(',').str[1]
df[['City', 'State']].head()
```

Out[326]:

	City	State
0	MOOSE CREEK	ID
1	BRIDGEPORT	CA
2	Saltville	VA
3	EUREKA	CA
4	Canton	OH

In [327]:
```
# checking the value count per country
df['Country'].value_counts()
```

Out[327]:
```
United States     82248
Brazil              374
Canada              359
Mexico              358
United Kingdom      344
                   ...
St Lucia              1
Albania               1
Palau                 1
Libya                 1
Niger                 1
Name: Country, Length: 219, dtype: int64
```

Observation: United states is responsible for 82248 entries out of 88888, 92.5% of the entire dataset. Hence analysing it individually would be instrumental

In [328]:
```
# Filter out NaN countries first
df_clean = df[df['Country'].notna()].copy()
df_US = df_clean[df_clean['Country'] == 'United States'].copy()
```

In [329]:
```
print(df_US.info())
print(df_US['Country'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 82248 entries, 0 to 88888
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype         
---  ------            --------------  -----         
 0   ID                82248 non-null  object        
 1   Type              82248 non-null  object        
 2   Date              82248 non-null  datetime64[ns]
 3   Location          82237 non-null  object        
 4   Country           82248 non-null  object        
 5   Injury_Severity   82140 non-null  object        
 6   Damage_type       80269 non-null  object        
 7   Make              82227 non-null  object        
 8   Model             82210 non-null  object        
 9   Engines           80373 non-null  float64       
 10  Engine_Type       79225 non-null  object        
 11  Flight_Purpose    79819 non-null  object        
 12  Fatal_Injuries    71594 non-null  float64       
 13  Serious_Injuries  70873 non-null  float64       
 14  Minor_Injuries    71519 non-null  float64       
 15  Uninjured         77243 non-null  float64       
 16  Weather           81603 non-null  object        
 17  Month             82248 non-null  int64         
 18  Season            82248 non-null  object        
 19  Year              82248 non-null  int64         
 20  City              82237 non-null  object        
 21  State             82237 non-null  object        
dtypes: datetime64[ns](1), float64(5), int64(2), object(14)
memory usage: 14.4+ MB
None
United States    82248
Name: Country, dtype: int64
```

In [330]:
```
# Damage_type

# Replacing UNK and Unk with Unknown
df['Damage_type'] = df['Damage_type'].str.title().replace({'Unk': 'Unknown'
```

In [331]:
```
# Weather

# fixing weather labels and replacing UNK with unknown
df['Weather'] = df['Weather'].str.upper().replace({'UNK': 'UNKNOWN'})
```

In [332]:
```
# Aircraft make
print(df['Make'].value_counts().head(10))
```

```
Cessna     22227
Piper      12029
CESSNA      4922
Beech       4330
PIPER       2841
Bell        2134
Boeing      1594
BOEING      1151
Grumman     1094
Mooney      1092
Name: Make, dtype: int64
```

In [333]:
```
# Convert to string, fill NaNs, and convert to uppercase for consistency

df['Make'] = df['Make'].astype(str).fillna('UNKNOWN').str.upper().str.strip
```

In [334]:
```
# MODEL
# Convert to string, fill NaNs, and convert to uppercase for consistency

df['Model'] = df['Model'].astype(str).fillna('UNKNOWN').str.upper().str.str
```

In [335]:
```
# Combining Make and Model for unique aircraft identification
df['Aircraft_Model'] = df['Make'] + ' ' + df['Model']
```

In [336]:
```
# ENGINE_TYPE
df['Engine_Type'].value_counts()
```

Out[336]:
```
Reciprocating      69530
Turbo Shaft         3609
Turbo Prop          3391
Turbo Fan           2481
Unknown             2051
Turbo Jet            703
None                  19
Geared Turbofan       12
Electric              10
LR                     2
NONE                   2
Hybrid Rocket          1
UNK                    1
Name: Engine_Type, dtype: int64
```

In [337]:
```
# ENGINE-TYPE
# Replacing 'NONE' with 'Unknown'
df['Engine_Type'] = df['Engine_Type'].replace('NONE', 'Unknown')

# Replacing other ambiguous categories with 'Unknown'
ambiguous_categories = ['None', 'UNK', 'LR']
df['Engine_Type'] = df['Engine_Type'].replace(ambiguous_categories, 'Unknow

# Standardizing naming
df['Engine_Type'] = df['Engine_Type'].replace({'Turbo Jet': 'Turbojet','Tur
    'Turbo Shaft': 'Turboshaft','Geared Turbofan': 'Turbofan', 'Hybrid Rock

# to verify if it worked
print(df['Engine_Type'].value_counts())
```

```
Reciprocating    69530
Turboshaft        3609
Turboprop         3391
Turbofan          2493
Unknown           2075
Turbojet           703
Electric            10
Hybrid               1
Name: Engine_Type, dtype: int64
```

In [338]:
```
# Convert injury counts to numeric
injury_cols = ['Fatal_Injuries', 'Serious_Injuries','Minor_Injuries', 'Unin
for col in injury_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce').fillna(0).astype(int)
```

In [339]:
```
# Removing the amount of injuries as this is aleady in another column
df['Injury_Severity'] = df['Injury_Severity'].str.split('(').str[0]
print(df['Injury_Severity'].value_counts())
```

```
Non-Fatal      67357
Fatal          17826
Incident        2219
Minor            218
Serious          173
Unavailable       96
Name: Injury_Severity, dtype: int64
```

In [340]:
```
# Create total injuries column
df['Total_Injuries'] = df['Fatal_Injuries'] + df['Serious_Injuries'] + df['
```

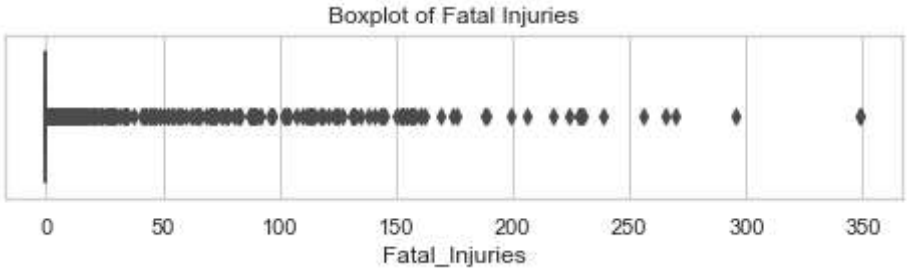
In [341]:
```
# checking all the columns we have now

df.columns
```

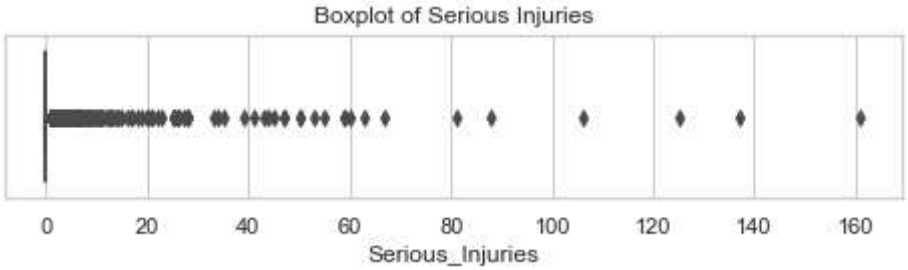
Out[341]:
```
Index(['ID', 'Type', 'Date', 'Location', 'Country', 'Injury_Severity',
       'Damage_type', 'Make', 'Model', 'Engines', 'Engine_Type',
       'Flight_Purpose', 'Fatal_Injuries', 'Serious_Injuries',
       'Minor_Injuries', 'Uninjured', 'Weather', 'Month', 'Season', 'Year',
       'City', 'State', 'Aircraft_Model', 'Total_Injuries'],
      dtype='object')
```

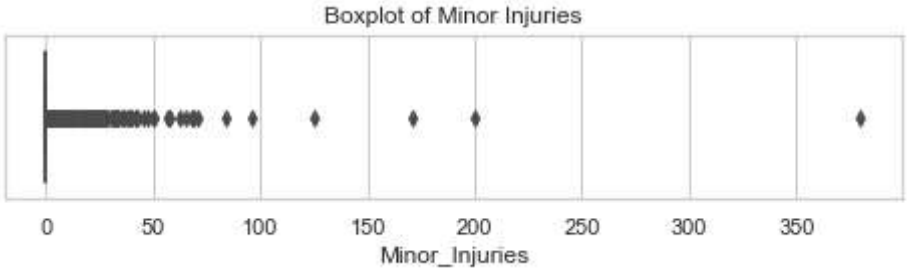
Univariate Analysis

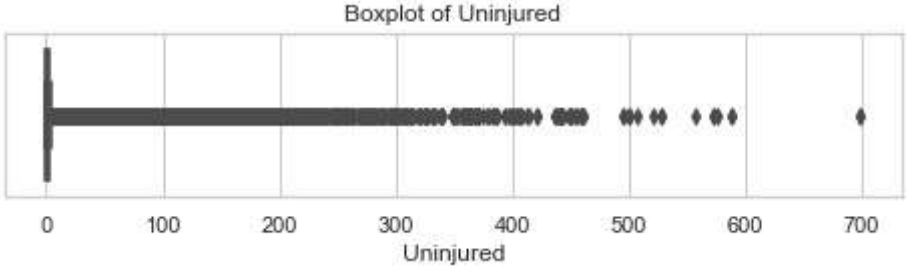
Numerical variables

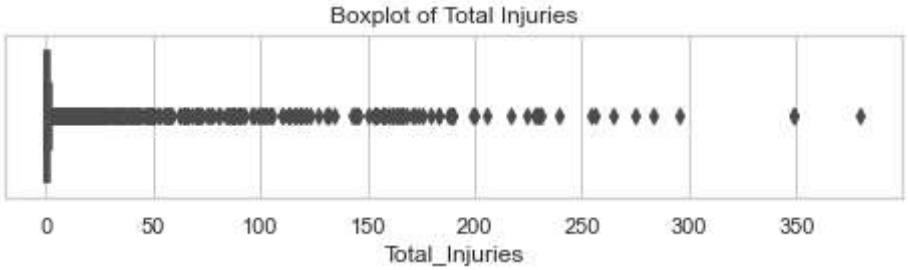
In [342]:
```
# BOXPLOT to detect outliers in each injury column
numeric_cols = ['Fatal_Injuries', 'Serious_Injuries', 'Minor_Injuries', 'Un

for col in numeric_cols:
    plt.figure(figsize=(8, 1.5))
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col.replace("_", " ")}')
    plt.show()

df[numeric_cols].describe().T
```











Out[342]:

	count	mean	std	min	25%	50%	75%	max
Fatal_Injuries	88889.0	0.564761	5.126649	0.0	0.0	0.0	0.0	349.0
Serious_Injuries	88889.0	0.240491	1.434614	0.0	0.0	0.0	0.0	161.0
Minor_Injuries	88889.0	0.309127	2.083715	0.0	0.0	0.0	0.0	380.0
Uninjured	88889.0	4.971245	27.002011	0.0	0.0	1.0	2.0	699.0
Total_Injuries	88889.0	1.114379	6.027319	0.0	0.0	0.0	1.0	380.0

Observation: Right-Skewed Distribution: the data is heavily right-skewed. Most data clusters near zero, but a few extreme values drag the tail to the right.

Categorical data

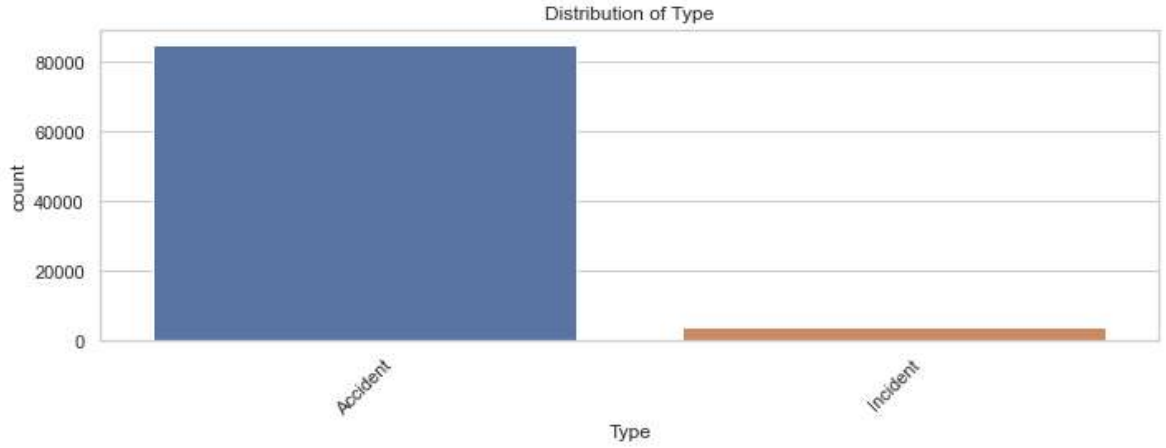
In [343]:

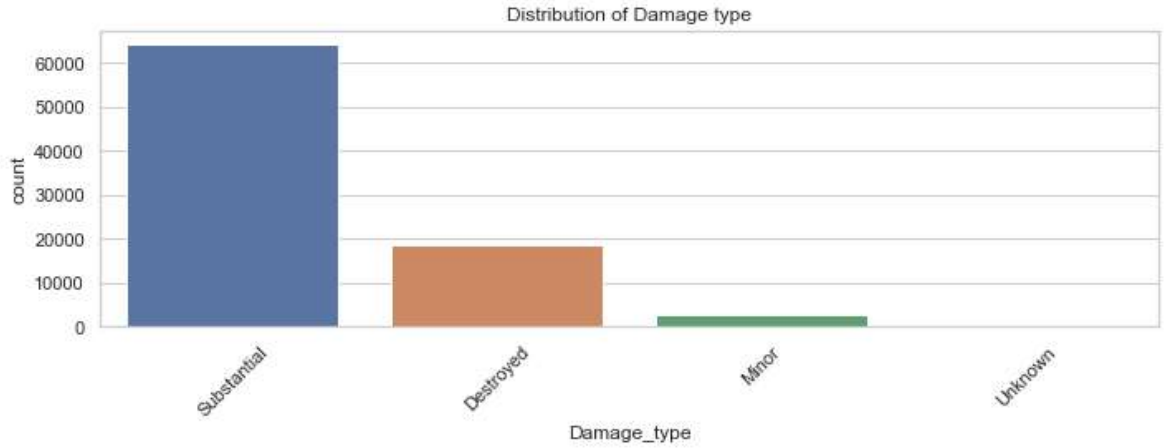
```
# Checking the categorical variables we have for our analysis
categorical_cols = df.select_dtypes(include='O').columns
print(categorical_cols)
```

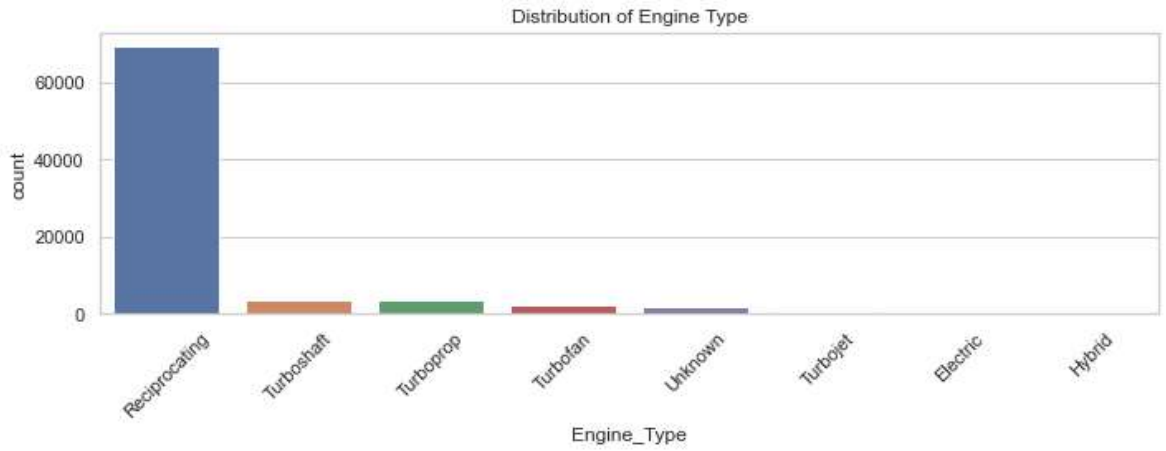
```
Index(['ID', 'Type', 'Location', 'Country', 'Injury_Severity', 'Damage_typ
e',
       'Make', 'Model', 'Engine_Type', 'Flight_Purpose', 'Weather', 'Seaso
n',
       'City', 'State', 'Aircraft_Model'],
      dtype='object')
```

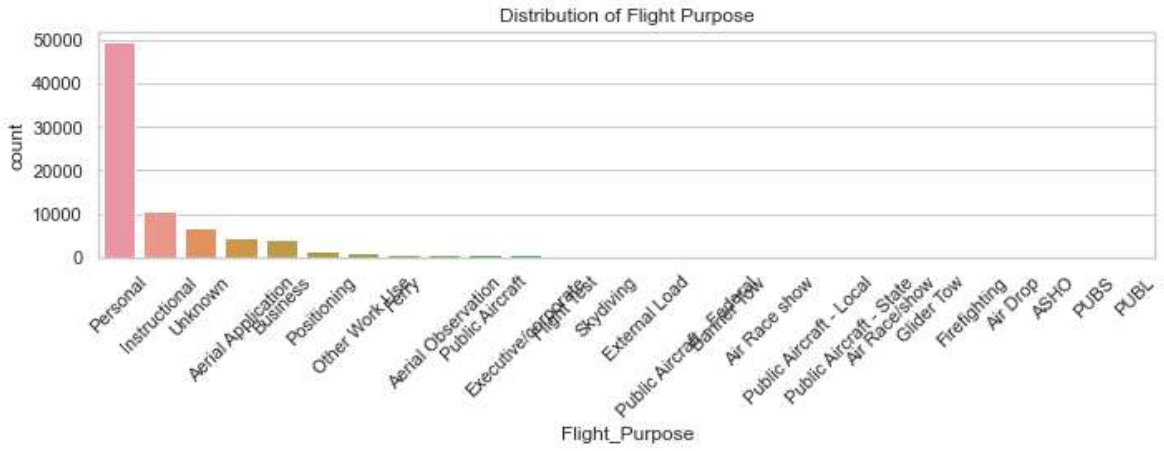
In [344]:

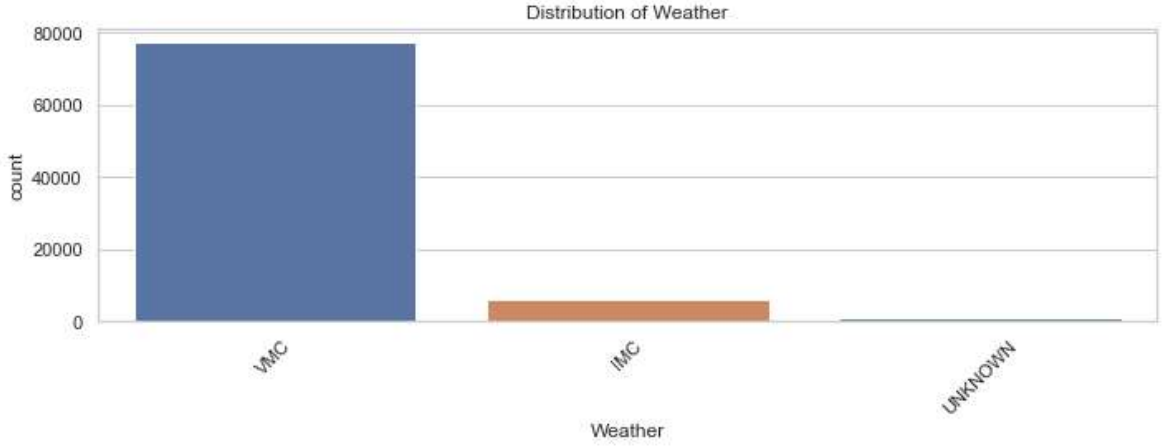
```
# Plotting a count plot for the key variables
for col in ['Type', 'Damage_type', 'Engine_Type', 'Flight_Purpose', 'Weathe
    plt.figure(figsize=(10, 4))
    sns.countplot(data=df, x=col, order=df[col].value_counts().index)
    plt.title(f'Distribution of {col.replace("_", " ")}')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

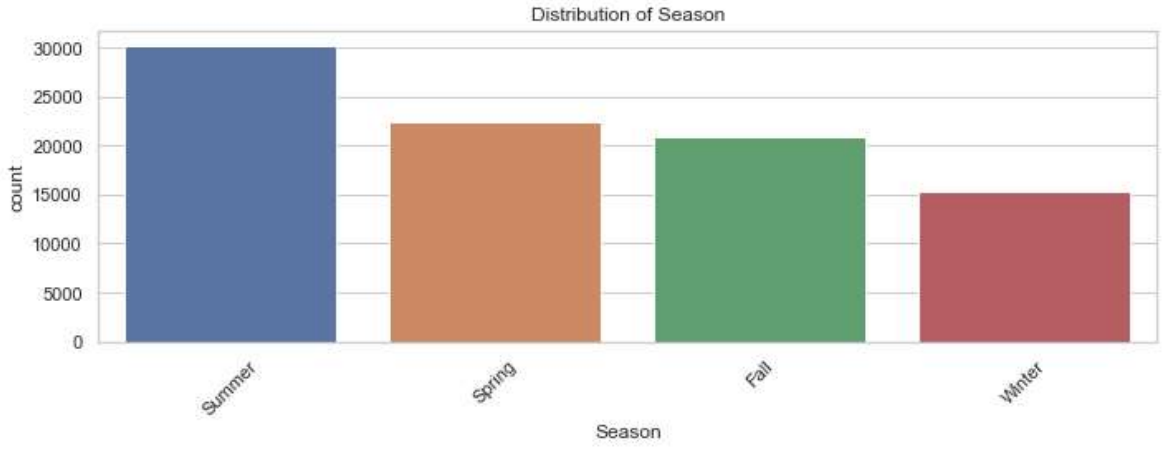






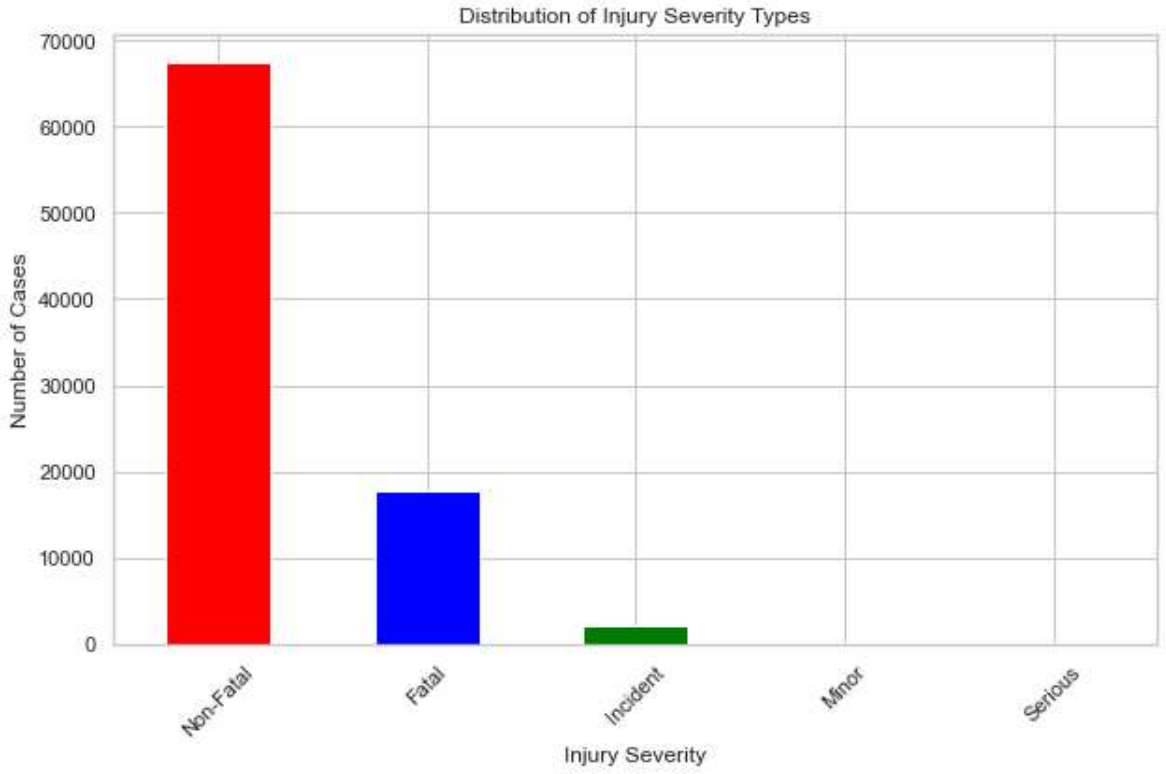

Distribution of Flight Purpose
count
50000
40000
30000
20000
10000
0
Personal
Instructional
Unknown
Aerial Application
Business
Positioning
Other Work Use
Ferry
Aerial Observation
Public Aircraft
Executive/corporate
Flight Test
Skydiving
External Load
Air Race show
Public Aircraft - Local
Public Aircraft - State
Air Race/show
Glider Tow
Firefighting
Air Drop
ASHO
PUBS
PUBL
Flight_Purpose


Distribution of Weather
count
80000
60000
40000
20000
0
VMC
IMC
UNKNOWN
Weather


Distribution of Season
count
30000
25000
20000
15000
10000
5000
0
Summer
Spring
Fall
Winter
Season

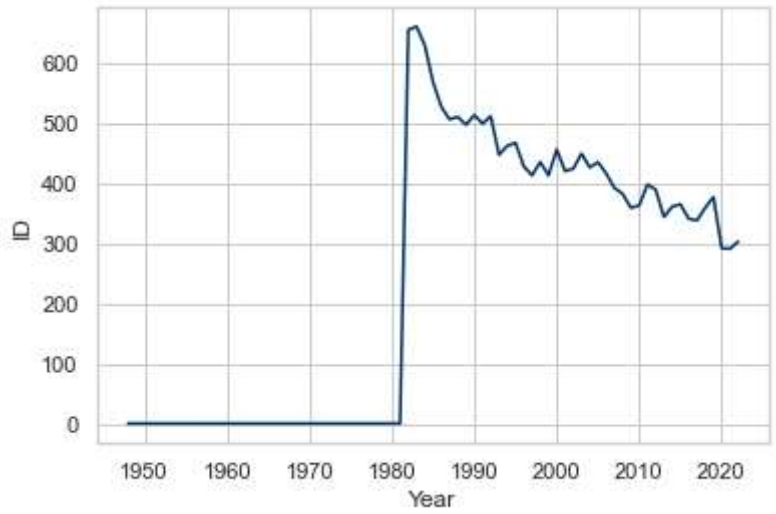
In [345]:

```
# Plotting the distribution of injury severity

# Filtering out 'Unavailable' entries
injury_df = df[df['Injury_Severity'] != 'Unavailable']
# value counts and sort
severity_counts = injury_df['Injury_Severity'].value_counts().sort_values(a

# Creating a bar plot
plt.figure(figsize=(10, 6))
severity_counts.plot(kind='bar', color=['red', 'blue', 'green', 'orange', '
plt.title('Distribution of Injury Severity Types')
plt.xlabel('Injury Severity')
plt.ylabel('Number of Cases')
plt.xticks(rotation=45)

plt.show()
```

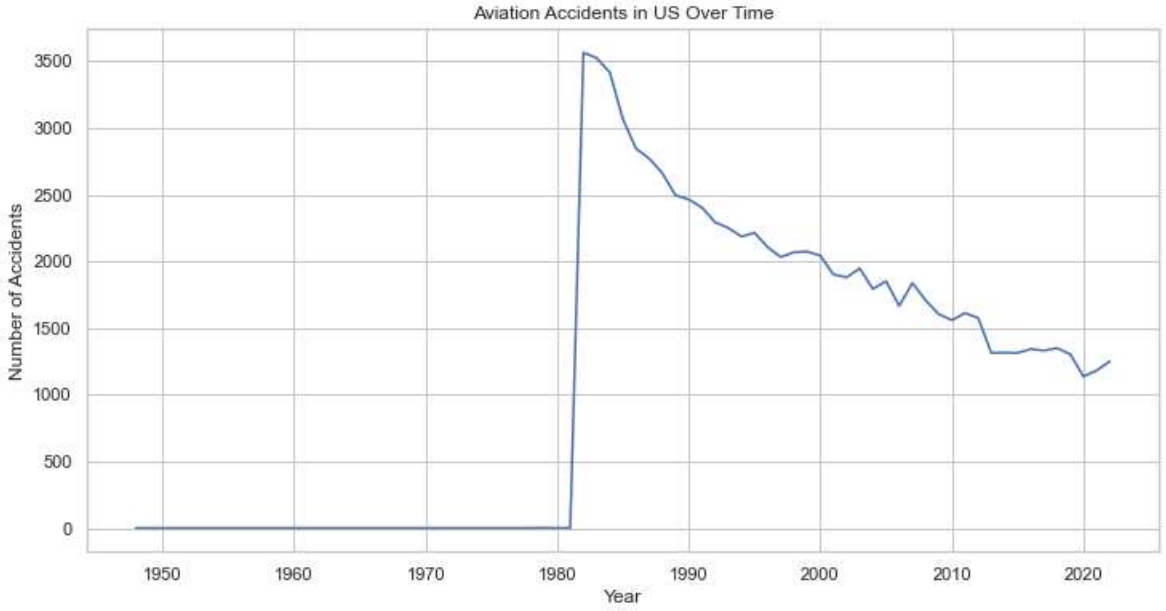


In [346]:
```
# Number of fatal accidents per year
fatal_accidents_per_year = df[df['Injury_Severity'] == 'Fatal'].groupby(['Y
sns.lineplot(x = 'Year', y = 'ID', data = fatal_accidents_per_year, color =
```

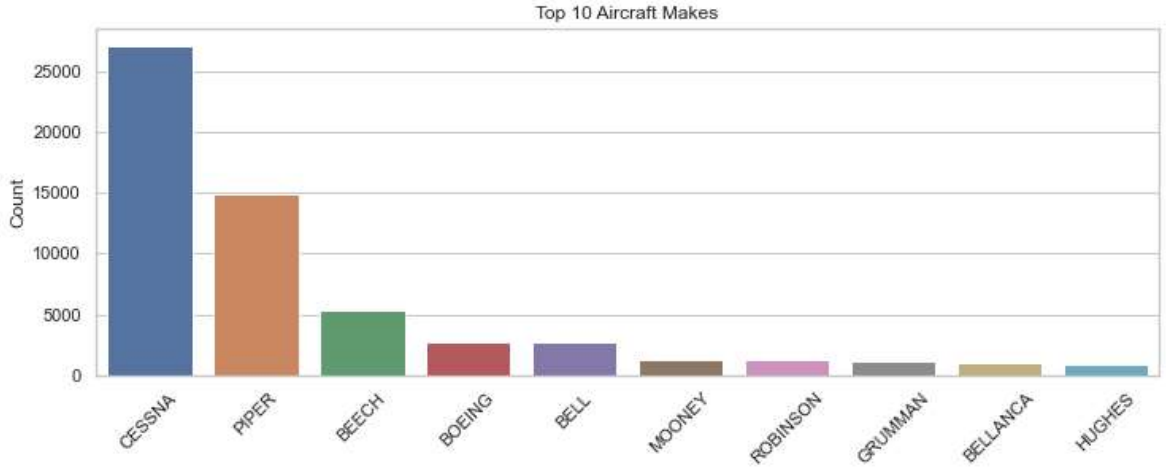
Out[346]: <AxesSubplot:xlabel='Year', ylabel='ID'>



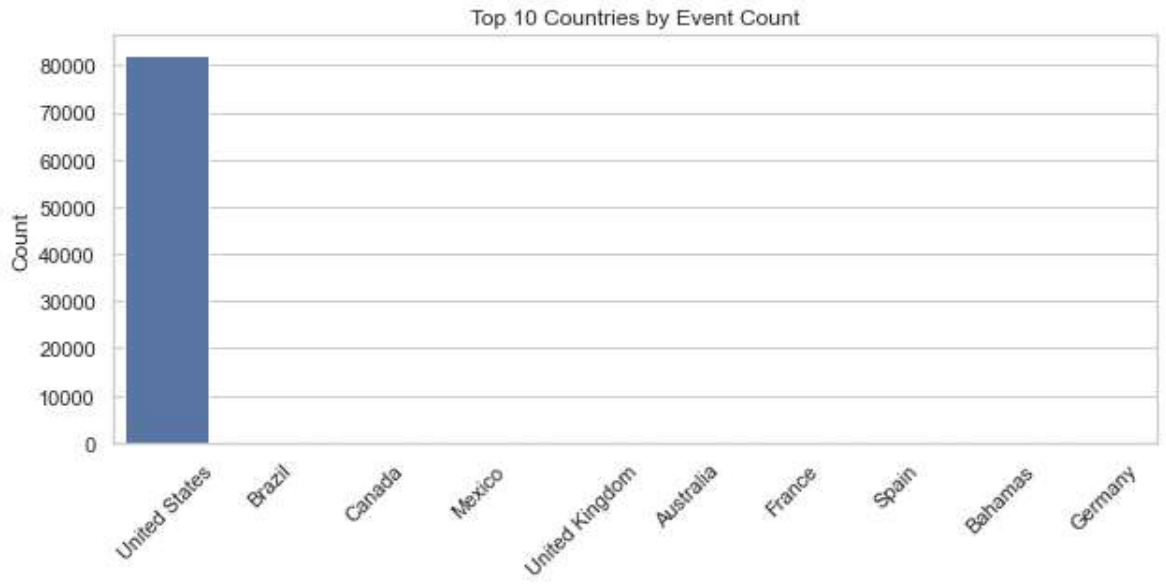
In [347]:
```
# Accidents trend in the US over time
plt.figure(figsize=(12,6))
df_US['Year'].value_counts().sort_index().plot()
plt.title('Aviation Accidents in US Over Time')
plt.xlabel('Year')
plt.ylabel('Number of Accidents')
plt.show()
```



In [348]:
```
# Plotting bar chart for the 10 most common aircraft make
plt.figure(figsize=(12, 4))
top_makes = df['Make'].value_counts().nlargest(10)
sns.barplot(x=top_makes.index, y=top_makes.values)
plt.title('Top 10 Aircraft Makes')
plt.xticks(rotation=45)
plt.ylabel('Count')
plt.show()
```



In [349]:
```
# Plotting a bar chat for the 10 most common countries
plt.figure(figsize=(10, 4))
top_countries = df['Country'].value_counts().nlargest(10)
sns.barplot(x=top_countries.index, y=top_countries.values)
plt.title('Top 10 Countries by Event Count')
plt.xticks(rotation=45)
plt.ylabel('Count')
plt.show()
```

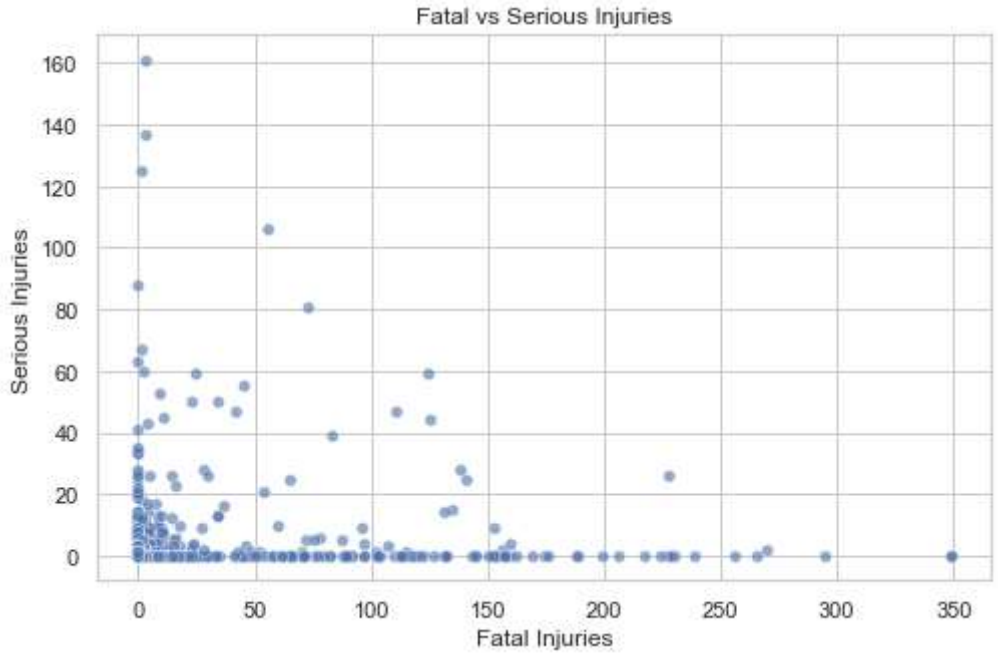


Observation: USA has the highest number of aviation events (accidents/incidents) recorded in the dataset, by over 80,000 events.

Bivariate Analysis

Numeric vs Numeric

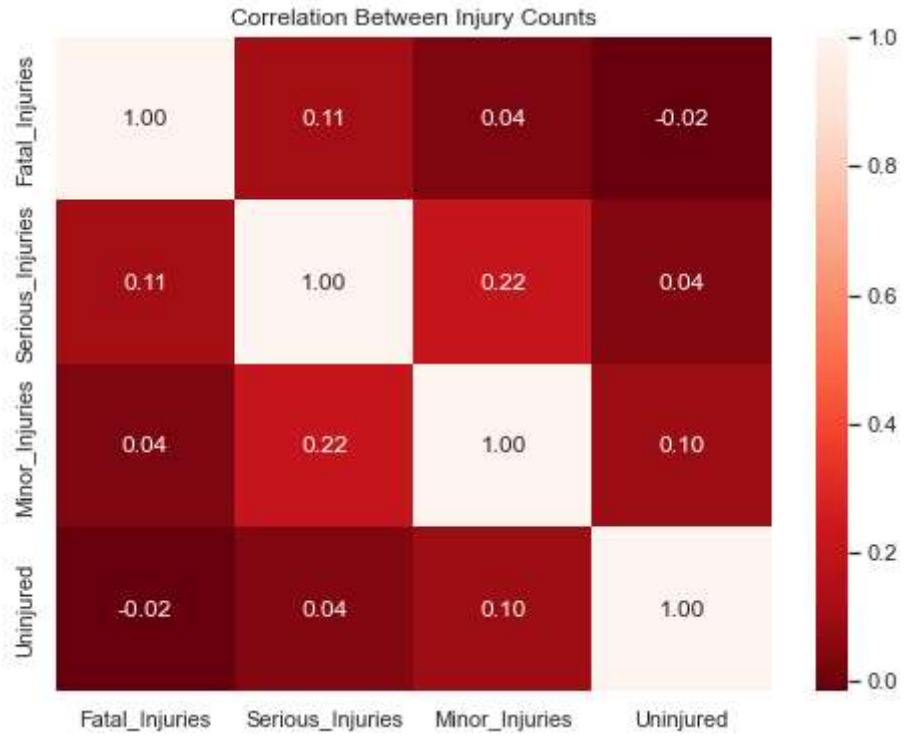
In [350]:
```
# to explore the correlations between numeric columns like injuries.
# scatter plot: Total Fatal Injuries vs Total Serious Injuries

plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x='Fatal_Injuries', y='Serious_Injuries', alpha=0.
plt.title('Fatal vs Serious Injuries')
plt.xlabel('Fatal Injuries')
plt.ylabel('Serious Injuries')
plt.show()

# Calculate the Pearson correlation coefficient
correlation = df[['Fatal_Injuries', 'Serious_Injuries']].corr()
print(correlation)
```




```
                 Fatal_Injuries  Serious_Injuries
Fatal_Injuries         1.000000          0.108066
Serious_Injuries       0.108066          1.000000
```

Observation: While both fatal and serious injuries can occur in aviation accidents, they don't strongly increase together. In fact, many events involve either serious or fatal injuries but not both, indicating differing accident severities and possibly different accident types or phases of flight. The correlation is approximately 0.1, which is very weak and positive.This confirms that fatal and serious injuries don’t strongly move together meaning one does not reliably predict the other.

In [351]:
```
# Correlation heatmap
numeric_cols = ['Fatal_Injuries', 'Serious_Injuries', 'Minor_Injuries', 'Un
plt.figure(figsize=(8, 6))
sns.heatmap(df[numeric_cols].corr(), annot=True, cmap='Reds_r', fmt=".2f")
plt.title('Correlation Between Injury Counts')
plt.show()
```

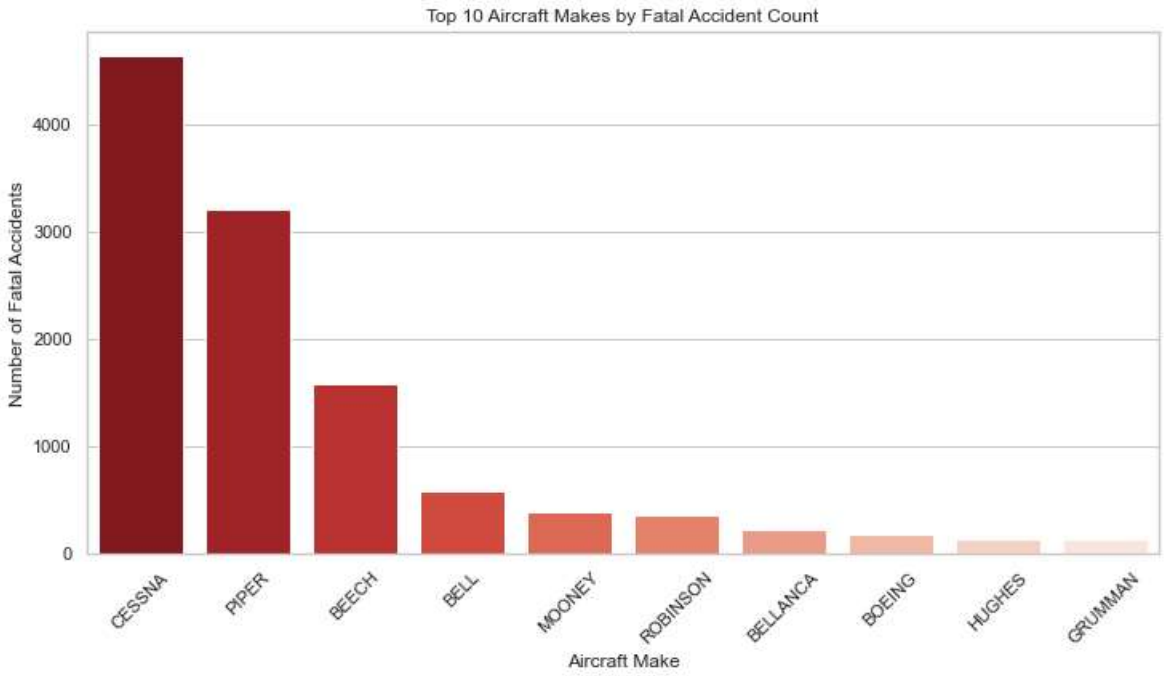


Observation: No strong linear relationship exists between any of the injury types.

Categorical vs Numerical

In [352]:

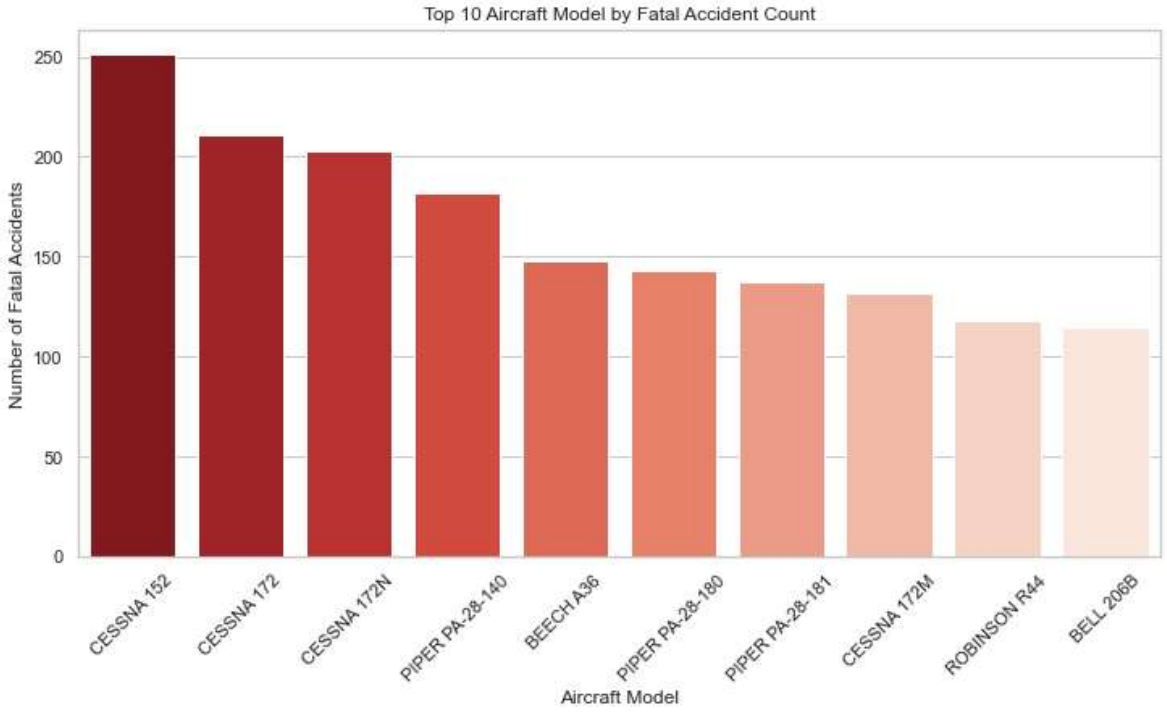
```
# Aircraft top makes and injury fatality
# Filter for fatal accidents and get top makes
fatal_df = df[df['Injury_Severity'] == 'Fatal']
top_makes = fatal_df['Make'].value_counts().head(10)

plt.figure(figsize=(12,6))
sns.barplot(x=top_makes.index, y=top_makes.values, palette='Reds_r')
plt.title('Top 10 Aircraft Makes by Fatal Accident Count')
plt.xlabel('Aircraft Make')
plt.ylabel('Number of Fatal Accidents')
plt.xticks(rotation=45)
plt.show()
```



Observation: Cessna is the ultimate leader in fatal accidents. *Recommendation:* Focus on pilot training and maintenance protocols for small aircraft to reduce fatalities.

In [353]:

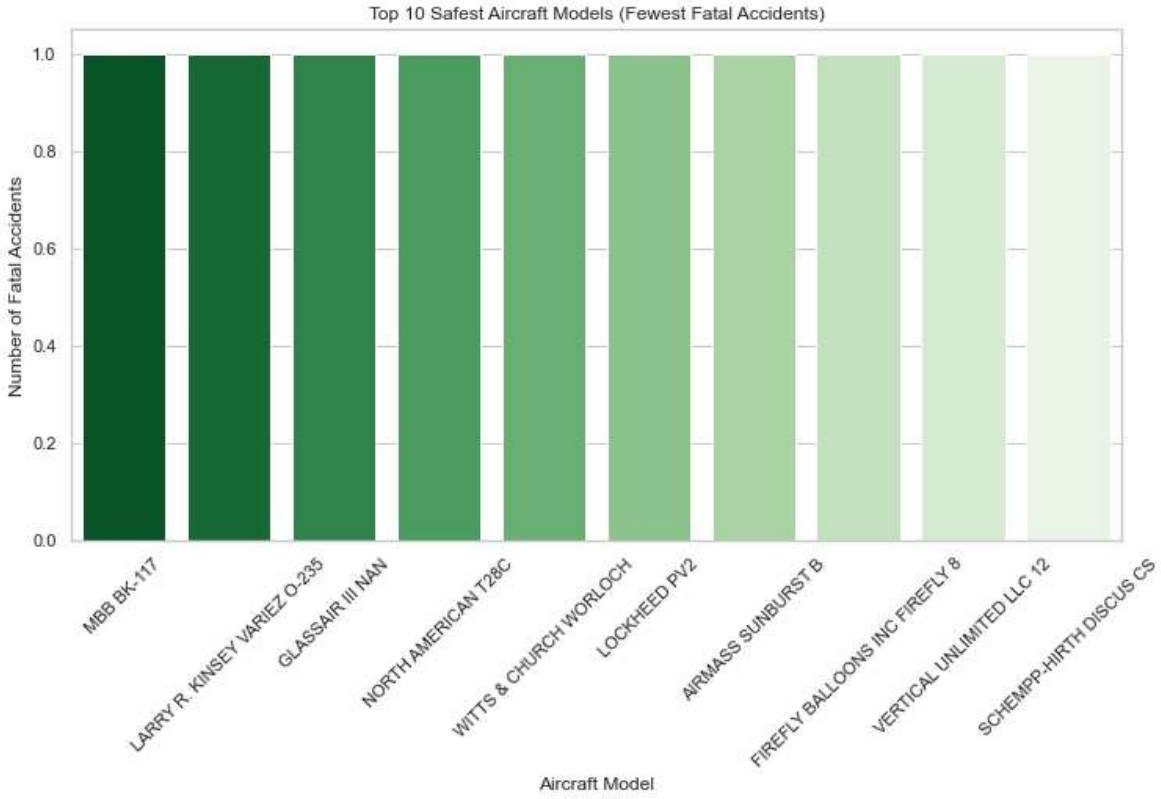
```
# Aircraft top model and injury fatality
# Filter for fatal accidents and get top model
fatal_df = df[df['Injury_Severity'] == 'Fatal']
top_model = fatal_df['Aircraft_Model'].value_counts().head(10)

plt.figure(figsize=(12,6))
sns.barplot(x=top_model.index, y=top_model.values, palette='Reds_r')
plt.title('Top 10 Aircraft Model by Fatal Accident Count')
plt.xlabel('Aircraft Model')
plt.ylabel('Number of Fatal Accidents')
plt.xticks(rotation=45)
plt.show()
```



Observation: Cessna 152, 172, and 172N are the three most risky aircraft model, due to the high fatality rate.

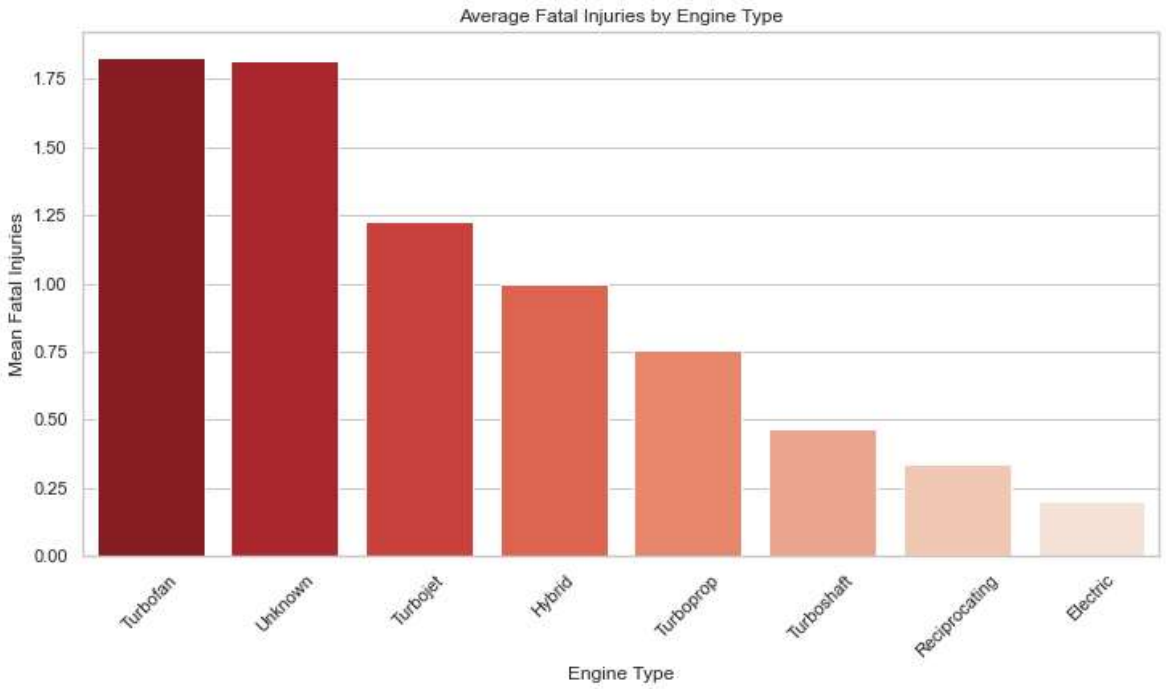
In [354]:

```
#  now lets check for the safest aircraft models

fatal_counts = df[df['Injury_Severity'] == 'Fatal']['Aircraft_Model'].value
safest_models = fatal_counts.nsmallest(10)  # Alternatively: .tail(10) if a

# Plot the results
plt.figure(figsize=(12, 6))
sns.barplot(x=safest_models.index, y=safest_models.values, palette='Greens_
plt.title('Top 10 Safest Aircraft Models (Fewest Fatal Accidents)')
plt.xlabel('Aircraft Model')
plt.ylabel('Number of Fatal Accidents')
plt.xticks(rotation=45)
plt.show()
```



Observation: The safest aircraft model barplot does not provide much insight into the analysis.

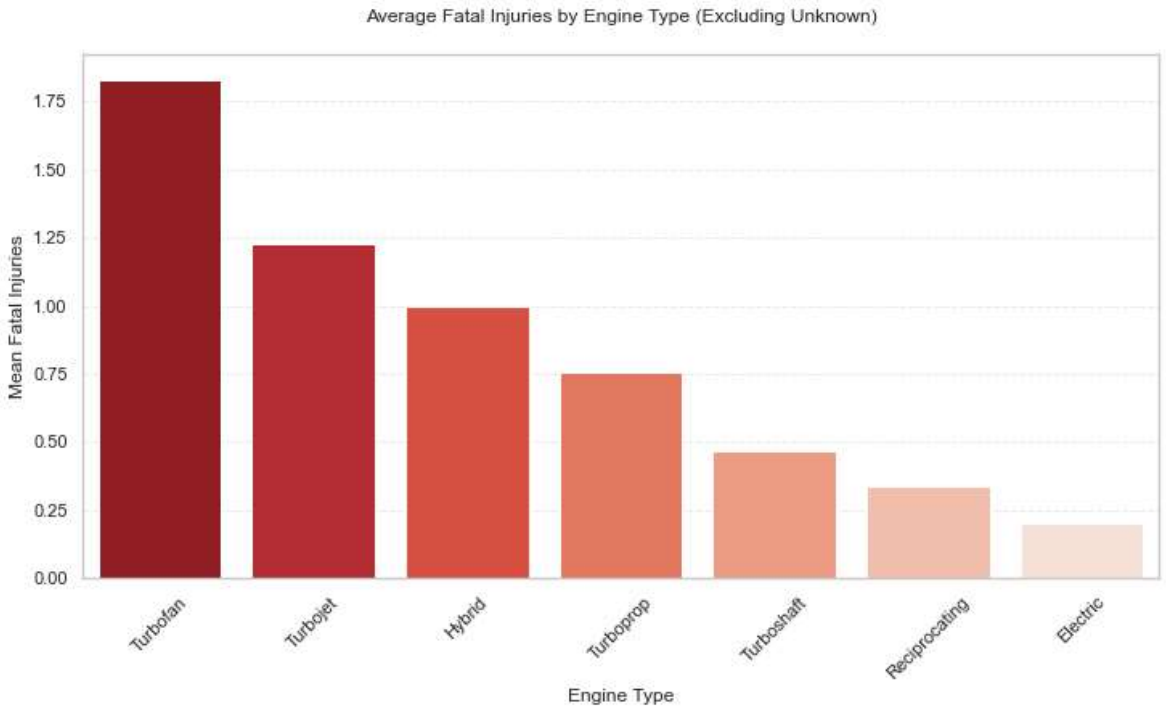
In [355]:
```
#  Average Fatal Injuries by Engine Type
avg_fatal = df.groupby('Engine_Type')['Fatal_Injuries'].mean().sort_values(

plt.figure(figsize=(12, 6))
sns.barplot(x=avg_fatal.index, y=avg_fatal.values, palette='Reds_r')
plt.title('Average Fatal Injuries by Engine Type')
plt.xlabel('Engine Type')
plt.ylabel('Mean Fatal Injuries')
plt.xticks(rotation=45)
plt.show()
```



Observation: The unknown data could be misleading in the analysis. Therefore, lets try excluding it to get an observation below:

In [356]:

```
# excluding the unknown engines
# Filter out 'Unknown' and calculate mean fatalities
engine_fatal = df[df['Engine_Type'] != 'Unknown'].groupby('Engine_Type')['F

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(
    x=engine_fatal.index,
    y=engine_fatal.values,
    palette='Reds_r',
    order=engine_fatal.index
)
plt.title('Average Fatal Injuries by Engine Type (Excluding Unknown)', pad=
plt.xlabel('Engine Type')
plt.ylabel('Mean Fatal Injuries')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.3)
plt.show()


# Calculate the number of engine types
count_engine = df[df['Engine_Type'] != 'Unknown']['Engine_Type'].value_coun
print(count_engine)
```




```
Reciprocating    69530
Turboshaft        3609
Turboprop         3391
Turbofan          2493
Turbojet           703
Electric            10
Hybrid               1
Name: Engine_Type, dtype: int64
```

Observation: Turbofan engines show the highest mean fatal injuries (1.75 per accident), indicating these incidents are more severe despite potentially fewer occurrences.

Recommendation: Focus on Turbofan safety by:

Investigating common failure modes (e.g., engine stalls, blade fractures).

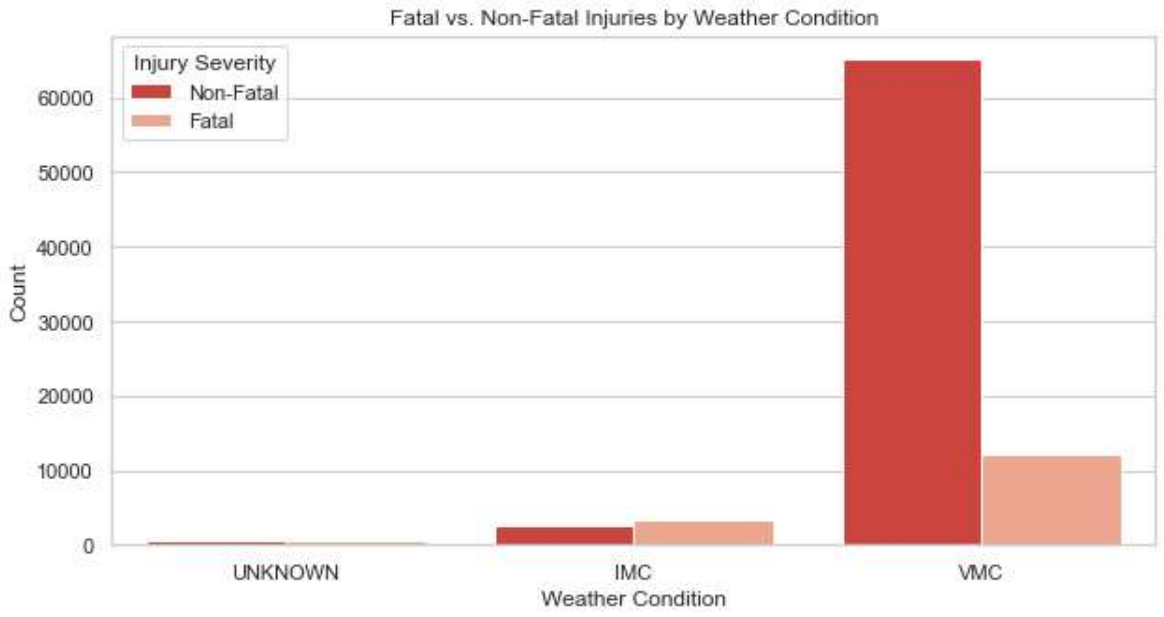
Enhancing pilot training for high-altitude emergencies.

Mandating stricter maintenance protocols for commercial fleets using these engines.

Categorical vs Categorical

In [357]:

```
#  Injury fatality by Weather
# Creating a new binary column (1 if fatal, 0 otherwise)
df['Is_Fatal'] = df['Fatal_Injuries'].apply(lambda x: 1 if x > 0 else 0)

# Plot
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='Weather', hue='Is_Fatal', palette='Reds_r')
plt.title('Fatal vs. Non-Fatal Injuries by Weather Condition')
plt.xlabel('Weather Condition')
plt.ylabel('Count')
plt.legend(title='Injury Severity', labels=['Non-Fatal', 'Fatal'])
plt.show()
```



Observation: VMC has more total fatalities due to higher accident volume. But to further confirm this, we need to check the rate of fatal injuries against the total injuries,as below:

In [358]:
```
# Calculating fatality rate (Fatal / Total Accidents) for each weather cond
weather_stats = df.groupby('Weather')['Is_Fatal'].agg(['count', 'mean'])
weather_stats.columns = ['Total_Accidents', 'Fatality_Rate']
weather_stats.sort_values('Fatality_Rate', ascending=False, inplace=True)
print(weather_stats)
```

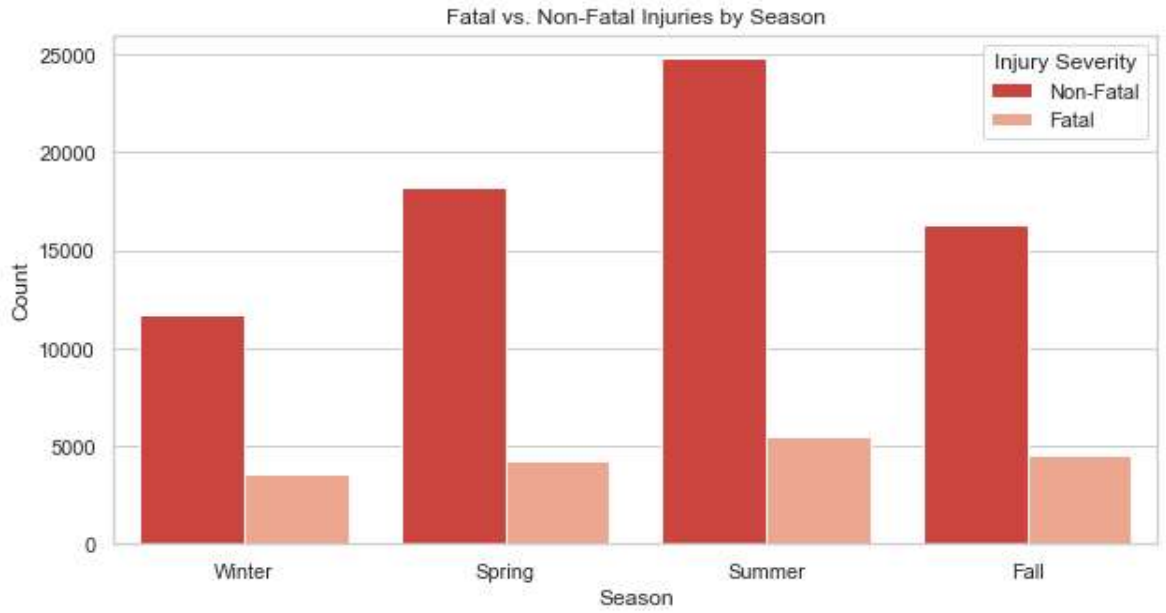
```
         Total_Accidents  Fatality_Rate
Weather
IMC                 5976       0.579485
UNKNOWN             1118       0.509839
VMC                77303       0.158584
```

Observation: IMC has fewer total accidents but a higher fatality rate (57.9% vs. VMC's 15.9%).

VMC has more accidents due to higher flight volume but a lower fatality rate.

Conclusion: IMC is riskier per accident, but VMC contributes more to total fatalities due to higher exposure.

In [359]:
```
# checking the fatality by season would give more insight

# Plotting fatalities by season
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='Season', hue='Is_Fatal', palette='Reds_r', order=
plt.title('Fatal vs. Non-Fatal Injuries by Season')
plt.xlabel('Season')
plt.ylabel('Count')
plt.legend(title='Injury Severity', labels=['Non-Fatal', 'Fatal'])
plt.show()

# fatality RATE by season (more insightful)
season_fatality = df.groupby('Season')['Is_Fatal'].mean().sort_values(ascen
print(season_fatality)
```




```
Season
Winter    0.233138
Fall      0.219304
Spring    0.187233
Summer    0.180526
Name: Is_Fatal, dtype: float64
```

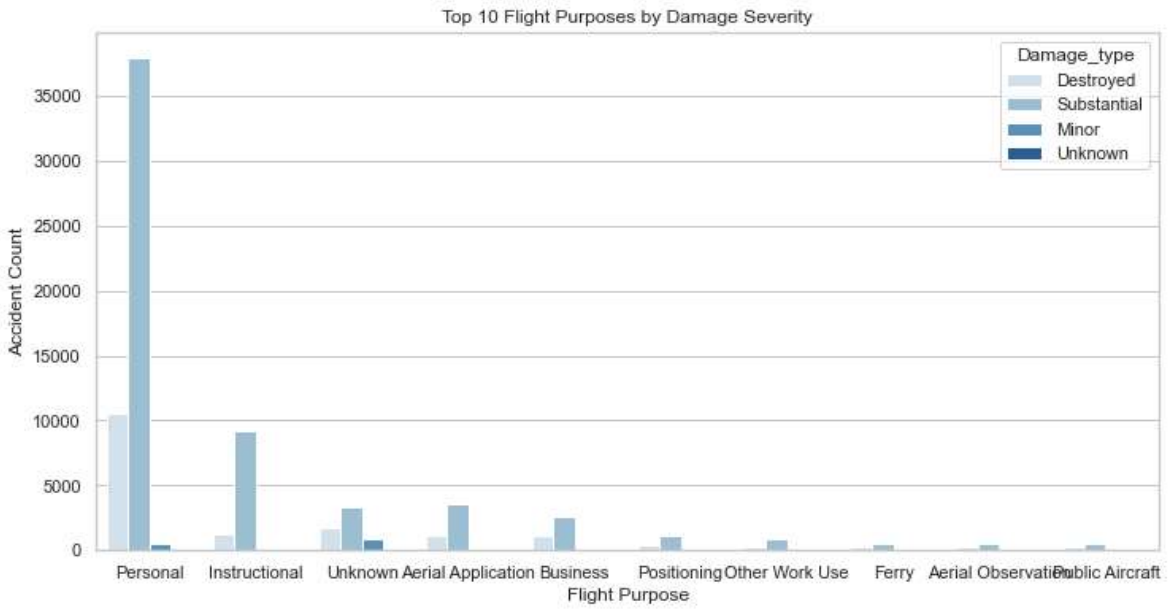
Observation: Winter has the highest fatality risk at 23%, followed by fall at 21%, spring 18% and summer is last at 18%, although it has the highest flights. This suggests weather conditions (ice, low visibility, turbulence) and operational challenges (de-icing, shorter daylight) increase accident severity.

Recommendation:

Flying in winter is riskier and focus should be on cold-weather emergencies pilot training.

In [360]:

```
#  flight purpose by damage type
# top 10 flight purposes by occurrence
top_purposes = df['Flight_Purpose'].value_counts().head(10).index

plt.figure(figsize=(12, 6))
sns.countplot(data=df[df['Flight_Purpose'].isin(top_purposes)],
    x='Flight_Purpose', hue='Damage_type', palette='Blues', order=top_purpo

plt.title('Top 10 Flight Purposes by Damage Severity')
plt.xlabel('Flight Purpose', fontsize=12)
plt.ylabel('Accident Count', fontsize=12)

plt.show()
```



Observation: Personal and instructional flights crash often; business flights crash rarely but catastrophically: target each with tailored safety measures.

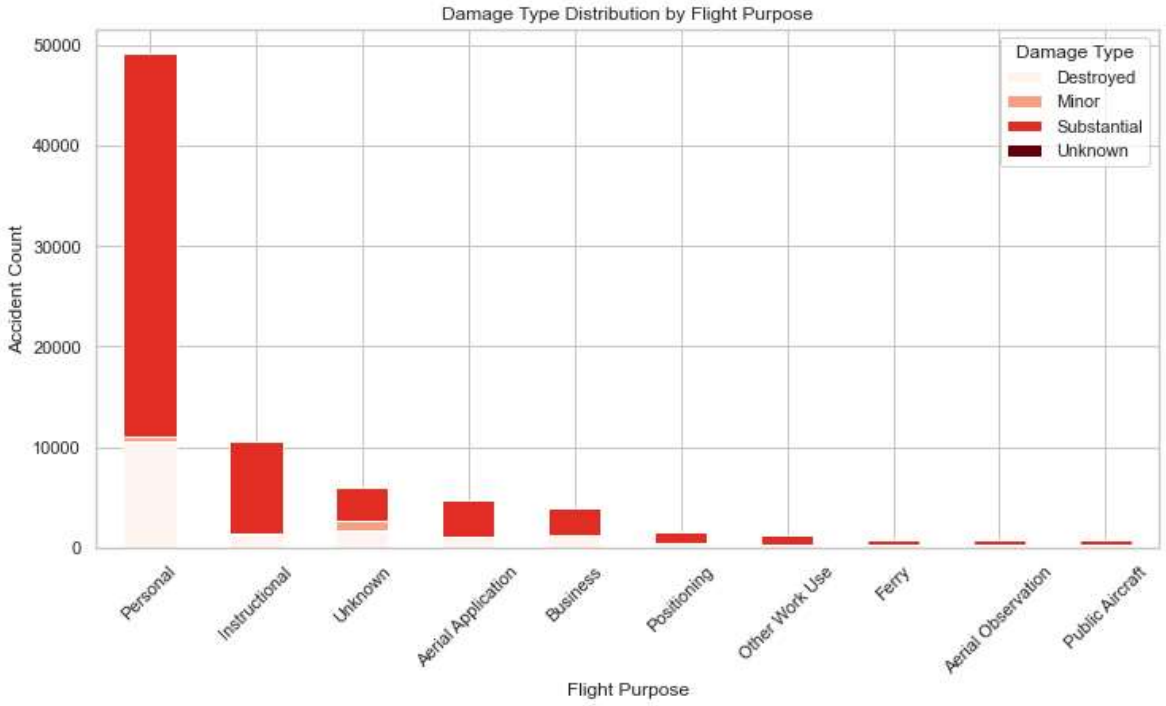
Multivariate Analysis

In [361]:
```
# Calculating crash frequency and mean fatalities per flight-purpose
purpose_stats = df.groupby('Flight_Purpose').agg(
    Total_Crashes=('Flight_Purpose', 'size'),
    Mean_Fatalities=('Fatal_Injuries', 'mean'),
    Destroyed_Rate=('Damage_type', lambda x: (x == 'Destroyed').mean())
).sort_values('Total_Crashes', ascending=False).head(10)

print(purpose_stats)
```

```
                    Total_Crashes  Mean_Fatalities  Destroyed_Rate
Flight_Purpose
Personal                    49448         0.379429        0.213558
Instructional               10601         0.180455        0.114329
Unknown                      6802         1.439136        0.256542
Aerial Application           4712         0.116511        0.226868
Business                     4018         0.575660        0.295172
Positioning                  1646         0.385784        0.250304
Other Work Use               1264         0.404272        0.210443
Ferry                         812         0.475369        0.289409
Aerial Observation            794         0.521411        0.283375
Public Aircraft               720         0.563889        0.290278
```

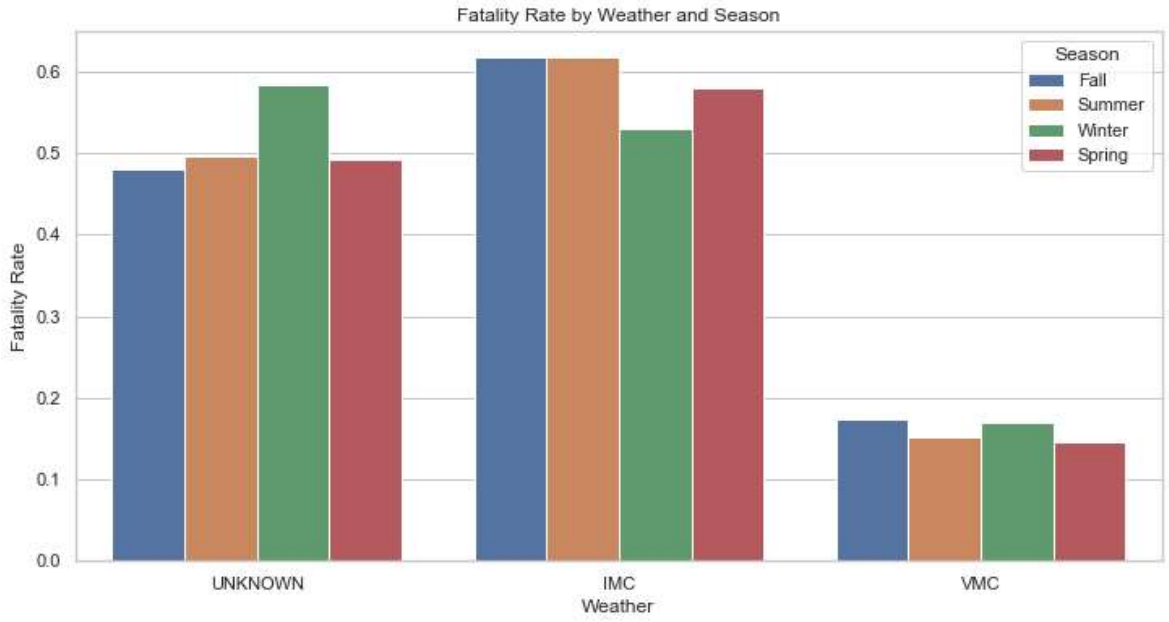
In [362]:

```
# Calculate damage type counts per purpose
damage_type = df.groupby(['Flight_Purpose', 'Damage_type']).size().unstack(
damage_type = damage_type.loc[purpose_stats.index]

# Plot
damage_type.plot(kind='bar', stacked=True, figsize=(12, 6), colormap='Reds'
plt.title('Damage Type Distribution by Flight Purpose')
plt.xlabel('Flight Purpose')
plt.ylabel('Accident Count')
plt.xticks(rotation=45)
plt.legend(title='Damage Type', bbox_to_anchor=(1, 1))
plt.show()
```



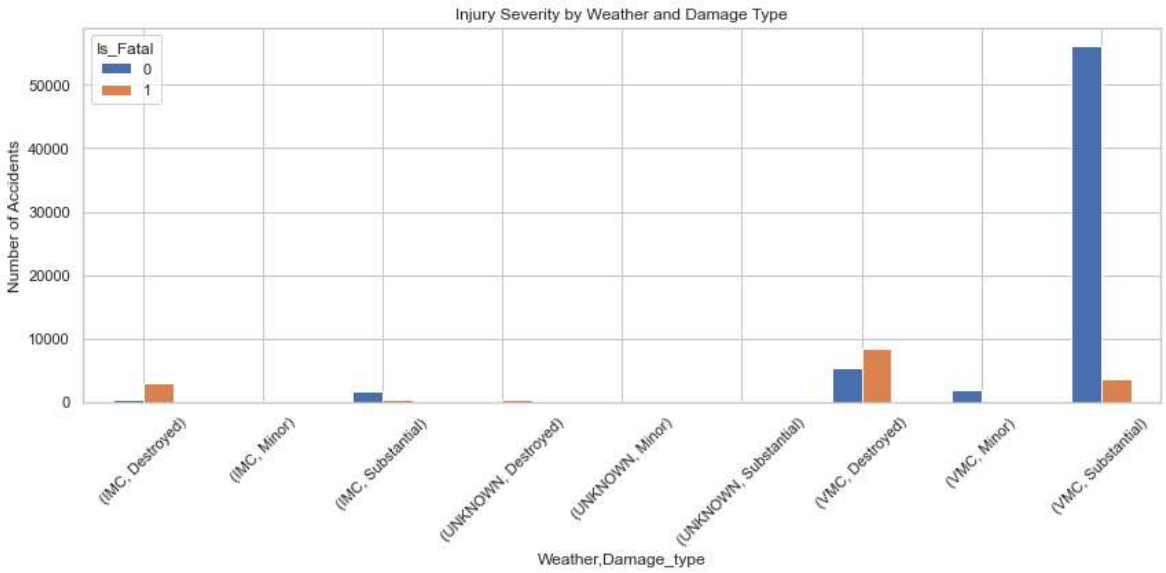
Observation: Business has the highest proportion of Destroyed (red segments). Personal dominates in volume but with mostly minor/substantial damage.

In [363]:
```
# Fatality rate by Weather AND Season
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Weather', y='Is_Fatal', hue='Season', estimator=np.
plt.title('Fatality Rate by Weather and Season')
plt.ylabel('Fatality Rate')
plt.show()
```



Observation: 1. Across all seasons, the fatality rate in IMC (poor visibility) is significantly higher than in VMC (clear weather). 2. The Winter-IMC bar is tallest than Winter-VMC, confirming cold + poor visibility is the deadliest combo. 3. Summer is the safest but still risky in IMC

In [364]:
```
# Crosstab/Grouped Bar Plot – Injury_Severity vs Weather + Damage_type

valid_damage = ['Destroyed', 'Substantial', 'Minor']
filtered_df = df[df['Damage_type'].isin(valid_damage)]
pd.crosstab(index=[filtered_df['Weather'], filtered_df['Damage_type']],
            columns=filtered_df['Is_Fatal']).plot(kind='bar', figsize=(12, 6

plt.title('Injury Severity by Weather and Damage Type')
plt.ylabel('Number of Accidents')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()


# Show fatality rates per group
crosstab = pd.crosstab(index=[df['Weather'], df['Damage_type']],
                       columns=df['Is_Fatal'], normalize='index')
print(crosstab.loc[:, 1])  # Fatality rates per group
```




```
Weather  Damage_type
IMC      Destroyed      0.879377
         Minor          0.011976
         Substantial    0.228622
UNKNOWN  Destroyed      0.869403
         Minor          0.020619
         Substantial    0.257534
         Unknown        0.250000
VMC      Destroyed      0.610708
         Minor          0.051785
         Substantial    0.060342
         Unknown        0.131579
Name: 1, dtype: float64
```

Observation:

Poor weather (IMC) and extensive damage (Destroyed) are strongly associated with fatal injuries. 87.9% fatality rate (IMC-Destroyed) vs. 61.1% (VMC-Destroyed).

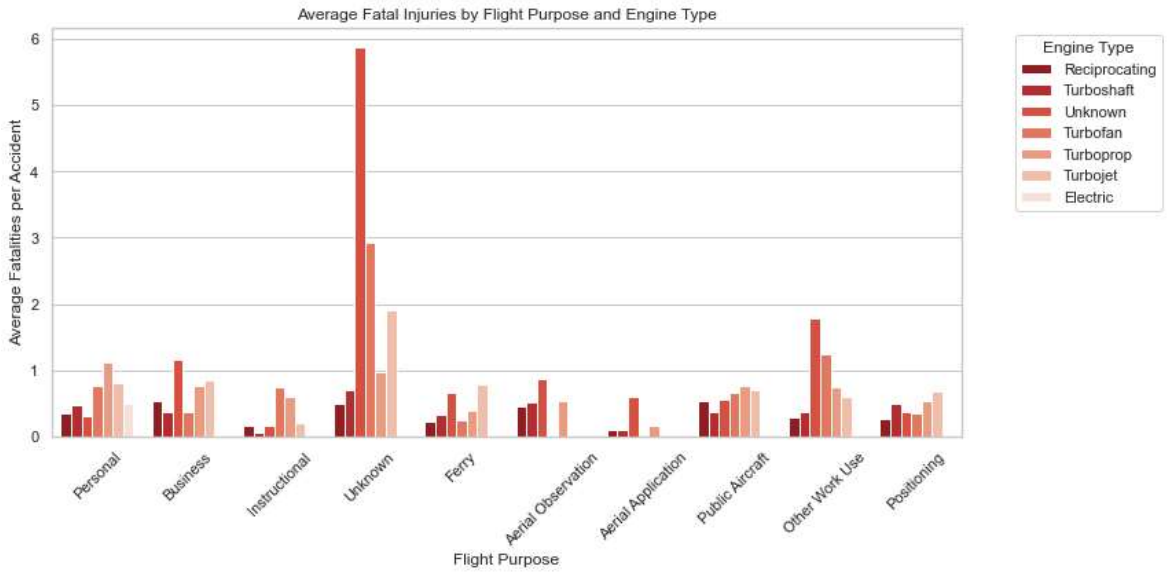
In [365]:
```
df.head(3)
```

Out[365]:

	ID	Type	Date	Location	Country	Injury_Severity	Damage_type	
0	20001218X45444	Accident	1948-10-24	MOOSE CREEK, ID	United States	Fatal	Destroyed	STI
1	20001218X45447	Accident	1962-07-19	BRIDGEPORT, CA	United States	Fatal	Destroyed	I
2	20061025X01555	Accident	1974-08-30	Saltville, VA	United States	Fatal	Destroyed	CE

3 rows × 25 columns

In [366]:
```
# Are commercial or private flights riskier? Does engine type matter?

# Top 10 purposes
top_purposes = df['Flight_Purpose'].value_counts().head(10).index
plot_data = df[df['Flight_Purpose'].isin(top_purposes)]

# Plotting
plt.figure(figsize=(12,6))
sns.barplot(data=plot_data,x='Flight_Purpose',
    y='Fatal_Injuries',
    hue='Engine_Type',
    estimator=np.mean,
    palette='Reds_r',
    errorbar=None)

plt.title('Average Fatal Injuries by Flight Purpose and Engine Type')
plt.xlabel('Flight Purpose')
plt.ylabel('Average Fatalities per Accident')
plt.xticks(rotation=45)
plt.legend(title='Engine Type', bbox_to_anchor=(1.05, 1))
plt.tight_layout()
plt.show()
```

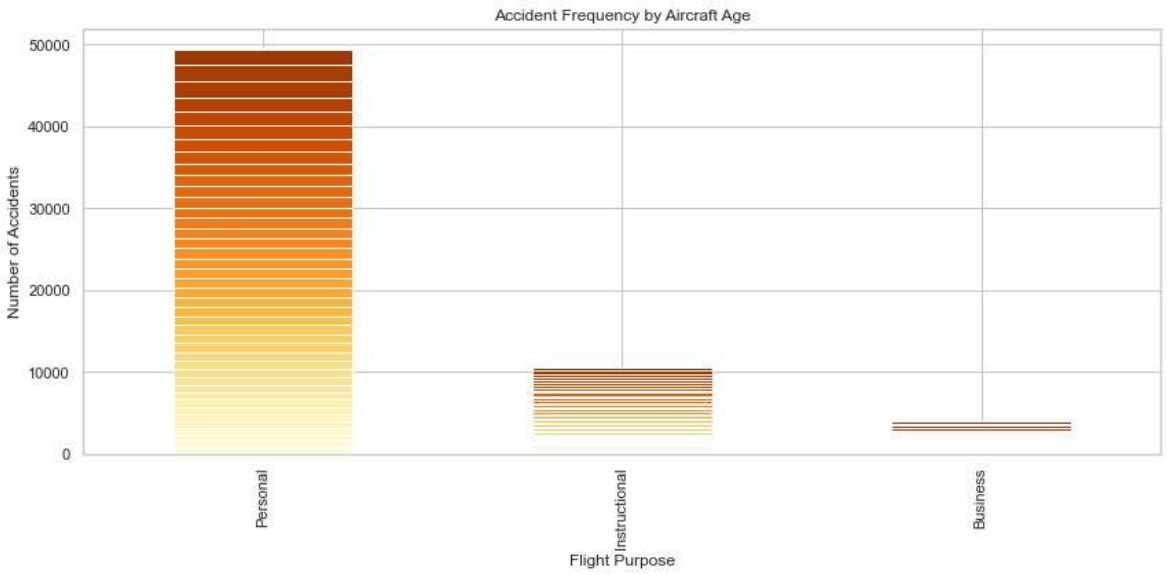


Observation:

Commercial-like flights (Business, Public) are riskier than private (Personal, Instructional), especially with jet/turboprop engines.

Engine type matters significantly: Turbojets/turboprops increase fatality rates 2-5x compared to reciprocating engines.

In [367]:
```
# Does aircraft age matter?
df['Aircraft_Age'] = 2023 - df['Year']

age_counts = df.groupby(['Flight_Purpose', 'Aircraft_Age']).size().unstack(

age_counts.loc[['Personal', 'Instructional', 'Business']].plot(
    kind='bar',
    stacked=True,
    figsize=(12, 6),
    color=sns.color_palette('YlOrBr', len(age_counts.columns)),
    legend=False)

plt.title('Accident Frequency by Aircraft Age')
plt.xlabel('Flight Purpose')
plt.ylabel('Number of Accidents')
plt.tight_layout()
plt.show()
```



In [368]:
```
# Saving the cleaned dataset to a CSV file
df.to_csv("df_cleaned.csv", index=False)
```