Introduction

This notebook examines aviation accident data to assess the risk of fatal and severe accidents across several key factors, including:

1. Flight Phases: Takeoff, landing, maneuvering, etc.
2. Weather Conditions: Impact of weather (e.g., rain, fog, wind) on accidents.
3. Aircraft and Engine Types: Model, make, engine types and number of engines.
4. Purpose of Flight: Commercial, private, training, etc.
5. Accident Severity: Categorized into fatal, serious, and minor injuries.

By analyzing these factors, the goal is to uncover trends and provide actionable recommendations for a company that is expanding into new industries to diversify its portfolio.

This analysis uses data from the National Transportation Safety Board (NTSB), covering civil aviation accidents from 1962 to 2023 in the United States and international waters.

Step 1: Import libraries

Firstly, we import all the necessary libraries required for data manipulation, analysis, and visualization. These libraries will help with data cleaning, statistical operations, and visual representation of results.

In [50]:

```
# --- Import libraries and loading dataset ---
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# --- Use 'ggplot' style for more visually appealing plots
plt.style.use('ggplot')
```

Step 2: Load the dataset

Here, we load the 'Aviation data' -a CSV file into a pandas DataFrame. This dataset contains the aviation accident data that we will analyze for risk patterns across various factors.

In [51]:

```
df = pd.read_csv('data\Aviation_Data.csv', low_memory=False)
df.head()
```

Out[51]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latituc
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	United States	Na
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	United States	Na
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	United States	36.9222
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	United States	Na

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latituc
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	United States	Na

5 rows × 31 columns

Step 3: Understand the structure of the dataset

After loading the dataset, it is important to perform a few basic checks to understand its structure, identify any missing values, and ensure that the data types are correct.

We will perform the following checks:

1. Shape of the dataset: To check the number of rows and columns.
2. Columns: To list all column names and ensure the dataset has the expected structure.
3. Preview the first 20 rows and check the content.
4. Data types: To confirm that numerical columns are correctly identified
5. Dataset summary: To get a quick overview of data types and missing values.
6. Missing values: To check if any columns contain missing values.

These checks will help us assess the quality of the dataset and prepare it for further analysis.

In [52]:
```
df.shape  # Returns (number of rows (90348), number of columns(31))
```

Out[52]:
```
(90348, 31)
```

In [53]:
```
df.columns # List all column names to inspect the dataset structure
```

Out[53]:
```
Index(['Event.Id', 'Investigation.Type', 'Accident.Number', 'Event.Date',
       'Location', 'Country', 'Latitude', 'Longitude', 'Airport.Code',
       'Airport.Name', 'Injury.Severity', 'Aircraft.damage',
       'Aircraft.Category', 'Registration.Number', 'Make', 'Model',
       'Amateur.Built', 'Number.of.Engines', 'Engine.Type', 'FAR.Description',
       'Schedule', 'Purpose.of.flight', 'Air.carrier', 'Total.Fatal.Injuries',
       'Total.Serious.Injuries', 'Total.Minor.Injuries', 'Total.Uninjured',
       'Weather.Condition', 'Broad.phase.of.flight', 'Report.Status',
       'Publication.Date'],
      dtype='object')
```

In [54]:
```
df.head(20) # Display the first 20 rows to preview the data
```

Out[54]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Lati
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	36.92
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	United States	
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	United States	

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Lati
5	20170710X52551	Accident	NYC79AA106	1979-09-17	BOSTON, MA	United States	42.44
6	20001218X45446	Accident	CHI81LA106	1981-08-01	COTTON, MN	United States	
7	20020909X01562	Accident	SEA82DA022	1982-01-01	PULLMAN, WA	United States	
8	20020909X01561	Accident	NYC82DA015	1982-01-01	EAST HANOVER, NJ	United States	
9	20020909X01560	Accident	MIA82DA029	1982-01-01	JACKSONVILLE, FL	United States	
10	20020909X01559	Accident	FTW82DA034	1982-01-01	HOBBS, NM	United States	
11	20020909X01558	Accident	ATL82DKJ10	1982-01-01	TUSKEGEE, AL	United States	
12	20020917X02148	Accident	FTW82FRJ07	1982-01-02	HOMER, LA	United States	
13	20020917X02134	Accident	FTW82FRA14	1982-01-02	HEARNE, TX	United States	
14	20020917X02119	Accident	FTW82FPJ10	1982-01-02	CHICKASHA, OK	United States	
15	20020917X02117	Accident	FTW82FPG08	1982-01-02	LITTLE ROCK, AR	United States	
16	20020917X01962	Accident	DEN82DTM08	1982-01-02	MIDWAY, UT	United States	
17	20020917X01656	Accident	ANC82FAG14	1982-01-02	SKWENTA, AK	United States	
18	20020917X02481	Accident	NYC82DA016	1982-01-02	GALETON, PA	United States	
19	20020917X02339	Accident	MIA82DA028	1982-01-02	MIAMI, FL	United States	

20 rows × 31 columns

In [55]:

```
df.info() # Dataset summary: data types, non-null counts, etc.
```

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

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

In [56]:
```
df.describe() # Summary statistics for numerical columns
```

Out[56]:

	Number.of.Engines	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries	Total.Uninjure
count	82805.000000	77488.000000	76379.000000	76956.000000	82977.00000
mean	1.146585	0.647855	0.279881	0.357061	5.32544
std	0.446510	5.485960	1.544084	2.235625	27.91363
min	0.000000	0.000000	0.000000	0.000000	0.00000
25%	1.000000	0.000000	0.000000	0.000000	0.00000
50%	1.000000	0.000000	0.000000	0.000000	1.00000
75%	1.000000	0.000000	0.000000	0.000000	2.00000
max	8.000000	349.000000	161.000000	380.000000	699.00000

In [57]:
```
df.dtypes # Check the data types of each column
```

Out[57]:
```
Event.Id                   object
Investigation.Type         object
Accident.Number            object
Event.Date                 object
Location                   object
Country                    object
Latitude                   object
Longitude                  object
Airport.Code               object
Airport.Name               object
Injury.Severity            object
Aircraft.damage            object
Aircraft.Category          object
Registration.Number        object
Make                       object
Model                      object
Amateur.Built              object
Number.of.Engines         float64
Engine.Type                object
FAR.Description            object
Schedule                   object
Purpose.of.flight          object
Air.carrier                object
```

```
Total.Fatal.Injuries      float64
Total.Serious.Injuries    float64
Total.Minor.Injuries      float64
Total.Uninjured           float64
Weather.Condition          object
Broad.phase.of.flight      object
Report.Status              object
Publication.Date           object
dtype: object
```

Step 4: Data cleaning (Handling missing values and duplicates)

We will:

1. Identify missing values: We will check which columns contain missing values and the number of missing entries.
2. Drop rows with missing `Event.Id` : Since `Event.Id` is the main unique identifier, it's crucial that this column has no missing values.
3. Check for duplicates: We will identify and remove any duplicate rows based on `Event.Id` , ensuring each event is represented only once.

These steps are necessary to clean the dataset before performing any further analysis.

In [58]:
```
# --- Check columns with missing values
df.isnull().sum().sort_values(ascending=False)  # List missing values per column

#Schedule, Air.carrier, and FAR.Description columns have the most missing values, wh
```

Out[58]:
```
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
Total.Serious.Injuries    13969
Total.Minor.Injuries      13392
Total.Fatal.Injuries      12860
Engine.Type                8536
Report.Status              7840
Purpose.of.flight          7651
Number.of.Engines          7543
Total.Uninjured            7371
Weather.Condition          5951
Aircraft.damage            4653
Registration.Number        2776
Injury.Severity            2459
Country                    1685
Amateur.Built              1561
Model                      1551
Make                       1522
Location                   1511
Event.Date                 1459
Accident.Number            1459
Event.Id                   1459
Investigation.Type            0
dtype: int64
```

In [59]:
```
# --- Check how many `Event.Id` row values are missing
df['Event.Id'].isna().sum()  # 1,459 missing values in the Event.Id column and we ne
```

Out[59]: 1459

In [60]:
```
# --- Drop rows where `Event.Id` is missing (since it's the main unique identifier)
df = df.dropna(subset=['Event.Id'])
```

In [61]:
```
# --- Confirm the shape of dataframe after dropping rows with missing `Event.Id`
df.shape  # # Returns (number of rows ((88889), number of columns(31))
```

Out[61]: (88889, 31)

In [62]:
```
# --- Check for duplicate `Event.Id` values
duplicate_ids = df['Event.Id'].duplicated().sum()    #938 duplicate values in the Ev
duplicate_ids
```

Out[62]: 938

In [63]:
```
# --- Remove duplicates present in the `Event.Id` column
df = df.drop_duplicates(subset=['Event.Id'])
```

In [64]:
```
# --- Confirm the new shape and uniqueness of `Event.Id` and the broader df
print(df.shape)   #the new rows are 87951 and 31 columns
print(df['Event.Id'].duplicated().sum())  #no duplicates in our unique identifier co
```

```
(87951, 31)
0
```

In [65]:
```
# -- Handling placeholders
placeholders = ['Unk', 'UNK', 'LR', 'N/A', 'na', 'nan', '-', 'None', 'NONE']
df = df.replace(placeholders, 'Unknown')

# Fill remaining missing values with 'unknown'
df = df.fillna('Unknown')
```

In [66]:
```
df.info()  # Check the current dataset structure (data types and non-null counts)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 87951 entries, 0 to 90347
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   Event.Id                87951 non-null  object
 1   Investigation.Type      87951 non-null  object
 2   Accident.Number         87951 non-null  object
 3   Event.Date              87951 non-null  object
 4   Location                87951 non-null  object
 5   Country                 87951 non-null  object
 6   Latitude                87951 non-null  object
 7   Longitude               87951 non-null  object
 8   Airport.Code            87951 non-null  object
 9   Airport.Name            87951 non-null  object
 10  Injury.Severity         87951 non-null  object
 11  Aircraft.damage         87951 non-null  object
 12  Aircraft.Category       87951 non-null  object
 13  Registration.Number     87951 non-null  object
 14  Make                    87951 non-null  object
 15  Model                   87951 non-null  object
 16  Amateur.Built           87951 non-null  object
 17  Number.of.Engines       87951 non-null  object
 18  Engine.Type             87951 non-null  object
 19  FAR.Description         87951 non-null  object
 20  Schedule                87951 non-null  object
```

```
 21  Purpose.of.flight       87951 non-null  object
 22  Air.carrier             87951 non-null  object
 23  Total.Fatal.Injuries    87951 non-null  object
 24  Total.Serious.Injuries  87951 non-null  object
 25  Total.Minor.Injuries    87951 non-null  object
 26  Total.Uninjured         87951 non-null  object
 27  Weather.Condition       87951 non-null  object
 28  Broad.phase.of.flight   87951 non-null  object
 29  Report.Status           87951 non-null  object
 30  Publication.Date        87951 non-null  object
dtypes: object(31)
memory usage: 21.5+ MB
```

Step 5: Data manipulation (Create a new DF copy with only useful columns)

This is meant to keep things simple for the analysis.

In [67]:

```
# -- Create a working copy with the most useful columns
use_cols = [
    'Event.Id', 'Event.Date','Make','Model','Aircraft.damage','Amateur.Built','Count
    'Number.of.Engines','Engine.Type','Purpose.of.flight',
    'Weather.Condition','Broad.phase.of.flight','Report.Status',
    'Total.Fatal.Injuries','Total.Serious.Injuries',
    'Total.Minor.Injuries','Total.Uninjured'
]

# -- Create the working dataframe
df_final = df[use_cols].copy()

# -- Look at what we kept-ie final dataset
df_final.shape
```

Out[67]: (87951, 17)

In [68]:

```
# -- Clean the event date column and create a “year” column
# Purpose is to ensure the date column is in datetime format and then extract the ye

# convert event.date to datetime; errors='coerce' will replace bad dates with NaT (m
df_final['Event.Date'] = pd.to_datetime(df_final['Event.Date'], errors='coerce')

# create a new column that contains only the year number (as integer)
df_final['year'] = df_final['Event.Date'].dt.year.astype('Int64')  # keeps NaN if an

# check to confirm it worked
df_final[['Event.Date', 'year']].head(10)
```

Out[68]:

	Event.Date	year
0	1948-10-24	1948
1	1962-07-19	1962
2	1974-08-30	1974
3	1977-06-19	1977
4	1979-08-02	1979
5	1979-09-17	1979
6	1981-08-01	1981
7	1982-01-01	1982

	Event.Date	year
8	1982-01-01	1982
9	1982-01-01	1982

In [69]:
```
# -- Clean a few text columns (eg make/model/purpose/weather/phase)
# -- Goal is to make sure text values like make, model, weather etc. are neat and ha


cols_to_clean = ['Make','Model','Country','Purpose.of.flight',
                 'Weather.Condition','Broad.phase.of.flight','Aircraft.damage']

for col in cols_to_clean:

    df_final[col] = df_final[col].astype(str)                                # convert
    df_final[col] = df_final[col].str.strip()                                # remove e
    df_final[col] = df_final[col].str.replace('  ', ' ')                    # replace doub
    df_final[col] = df_final[col].str.upper()                                # convert

df_final[['Make','Model','Purpose.of.flight','Weather.Condition','Broad.phase.of.fli
```

Out[69]:

	Make	Model	Purpose.of.flight	Weather.Condition	Broad.phase.of.flight
0	STINSON	108-3	PERSONAL	UNKNOWN	CRUISE
1	PIPER	PA24-180	PERSONAL	UNKNOWN	UNKNOWN
2	CESSNA	172M	PERSONAL	IMC	CRUISE
3	ROCKWELL	112	PERSONAL	IMC	CRUISE
4	CESSNA	501	PERSONAL	VMC	APPROACH
5	MCDONNELL DOUGLAS	DC9	UNKNOWN	VMC	CLIMB
6	CESSNA	180	PERSONAL	IMC	UNKNOWN
7	CESSNA	140	PERSONAL	VMC	TAKEOFF
8	CESSNA	401B	BUSINESS	IMC	LANDING
9	NORTH AMERICAN	NAVION L-17B	PERSONAL	IMC	CRUISE

Step 6: Exploring general patterns

Here we will try to understand how aviation accidents have changed over time and to identify any clear patterns in the data, highlighting how accident frequency and severity have evolved

We will start by analyzing the number of accidents per year, then look at additional high-level patterns such as fatality rates, purpose of flight, weather conditions, and aircraft make.

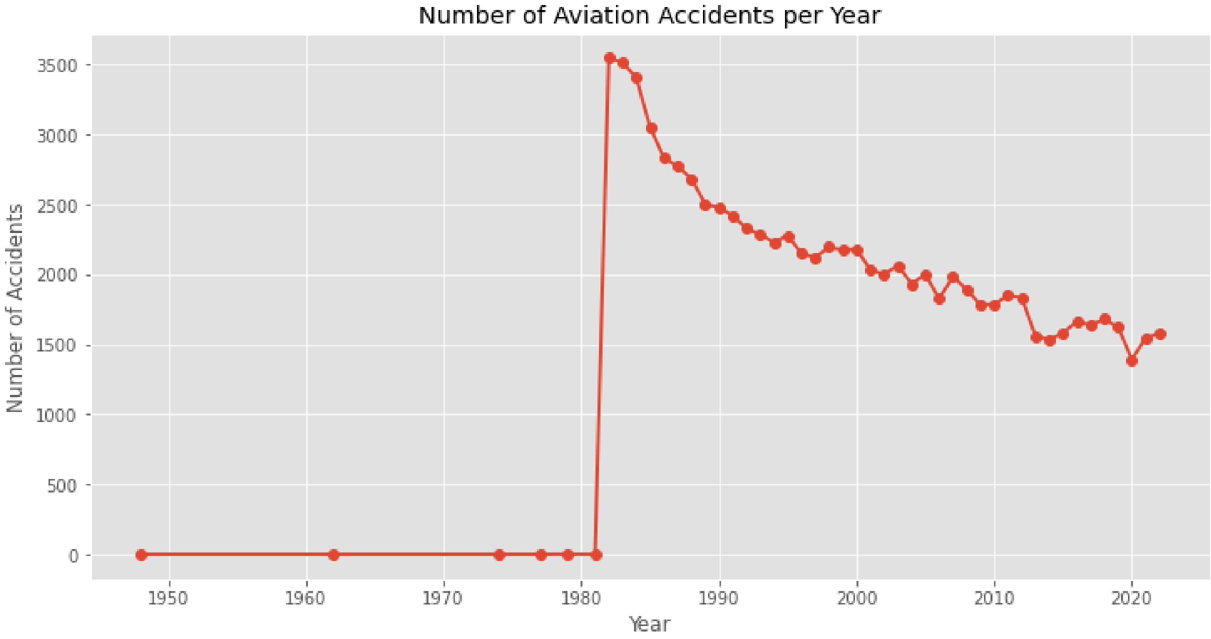
In [70]:
```
# -- Number of accidents each year


accidents_per_year = df_final['year'].value_counts().sort_index()
accidents_per_year
```

Out[70]:
```
1948        1
1962        1
1974        1
```

```
1977       1
1979       2
1981       1
1982    3547
1983    3513
1984    3406
1985    3053
1986    2832
1987    2773
1988    2685
1989    2502
1990    2480
1991    2420
1992    2328
1993    2285
1994    2229
1995    2278
1996    2150
1997    2121
1998    2196
1999    2174
2000    2183
2001    2032
2002    2001
2003    2063
2004    1932
2005    2001
2006    1826
2007    1984
2008    1893
2009    1783
2010    1786
2011    1850
2012    1835
2013    1561
2014    1535
2015    1582
2016    1664
2017    1638
2018    1681
2019    1624
2020    1392
2021    1545
2022    1581
Name: year, dtype: Int64
```

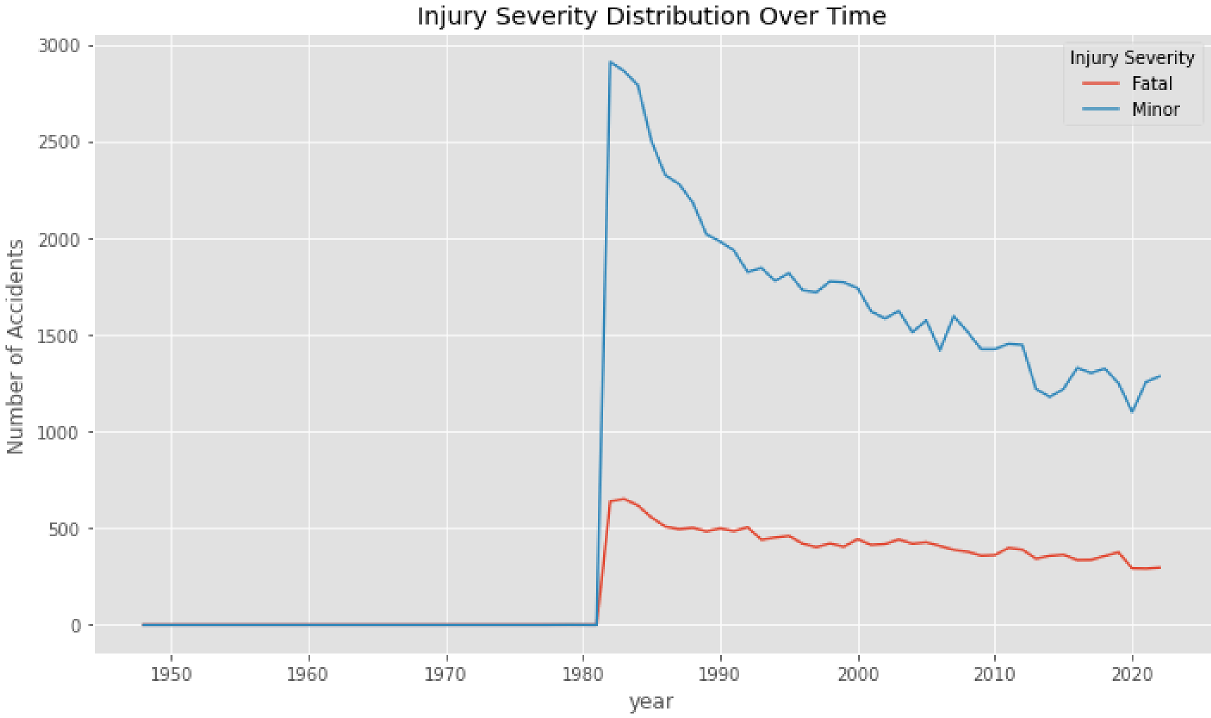
In [71]:

```python
# -- plot
accidents_per_year = (
    df_final.dropna(subset=['year'])
            .groupby('year')['Event.Id'].count()
            .reset_index()
)
accidents_per_year['year'] = accidents_per_year['year'].astype(int)

plt.figure(figsize=(12,6))
plt.plot(accidents_per_year['year'], accidents_per_year['Event.Id'], marker='o', lin
plt.title('Number of Aviation Accidents per Year')
plt.xlabel('Year'); plt.ylabel('Number of Accidents'); plt.grid(True)
plt.show()

#The graph confirms that:
# 1. The number of accidents was very high in the 1980s, exceeding 3,000 accidents p
# 2. Accident frequency has halved since the 1980s.
# 3. Aviation has become progressively safer, even as global air traffic has increas
# The analysis will therefore focus on data from 1982 onward, where records are more
```



In [72]:

```
#  -- Plot on accident severity (Fatal vs. Non-Fatal) over time

# Convert 'Total.Fatal.Injuries' to numeric, forcing errors to NaN
df_final['Total.Fatal.Injuries'] = pd.to_numeric(df_final['Total.Fatal.Injuries'], e

df_final['Injury Severity'] = df_final['Total.Fatal.Injuries'].apply(
    lambda x: 'Fatal' if x > 0 else ('Serious' if x > 0 else 'Minor'))

severity_by_year = df_final.groupby(['year', 'Injury Severity']).size().unstack().fi

severity_by_year.plot(kind='line', figsize=(10, 6))        # Plotting the injury sev
plt.title('Injury Severity Distribution Over Time')
plt.xlabel('year')
plt.ylabel('Number of Accidents')
plt.legend(title='Injury Severity')
plt.grid(True)
plt.tight_layout()
plt.show()

# 1. Fatal accidents peaked around the 1980s and have significantly reduced since th
# 2. Minor accidents have followed a similar downward trend, but with a less pronoun
# 3. Both fatal and minor accidents have shown a steady decline since the 1980s, ref
```



Step 7: Data analysis and visualization

We are now going to create two new columns that will make it easy to measure accident severity:

- `total_injuries` – total people hurt in an event.
- `is_fatal` – flag if anyone died (1 = yes).
- `is_severe` – flag if fatal or serious injuries occurred.

With the above we can finally, calculate **overall fatal and severe accident rates** against key factors such as; Phase of flight, Weather condition, Make and model of aircraft, Engine type, Purpose of flight, Number of engines

In [73]:

```
# -- Creating two new columns that are necessary to help us measure accident severit

# -- 1. Total_injuries - Total number of people hurt in that accident. It adds up al
# -- 2. is_fatal - shows whether anyone died in the accident. It checks if Total.Fat
      #-> 1 means yes, at least one person died.
      #-> 0 means no deaths occurred.
# -- 3. is_severe - shows whether the accident was serious (someone died or had seri
      #-> 1 means the accident was severe.
      #-> 0 means there were no serious injuries or deaths.

injury_columns = ['Total.Fatal.Injuries', 'Total.Serious.Injuries', 'Total.Minor.Inj

for column in injury_columns:
    df_final[column] = pd.to_numeric(df_final[column], errors='coerce').fillna(0)

# create helper flags
# 'is_fatal' = 1 if any fatalities happened, else 0
# 'is_severe' = 1 if there were fatalities or serious injuries, else 0

df_final['is_fatal'] = (df_final['Total.Fatal.Injuries'] > 0).astype(int)
df_final['is_severe'] = (
    (df_final['Total.Fatal.Injuries'] + df_final['Total.Serious.Injuries']) > 0
).astype(int)
```

```
# quick check to confirm
df_final[['Total.Fatal.Injuries', 'Total.Serious.Injuries', 'is_fatal', 'is_severe']]
```

Out[73]:

	Total.Fatal.Injuries	Total.Serious.Injuries	is_fatal	is_severe
0	2.0	0.0	1	1
1	4.0	0.0	1	1
2	3.0	0.0	1	1
3	2.0	0.0	1	1
4	1.0	2.0	1	1

In [74]:

```
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 87951 entries, 0 to 90347
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Event.Id                87951 non-null  object
 1   Event.Date              87951 non-null  datetime64[ns]
 2   Make                    87951 non-null  object
 3   Model                   87951 non-null  object
 4   Aircraft.damage         87951 non-null  object
 5   Amateur.Built           87951 non-null  object
 6   Country                 87951 non-null  object
 7   Number.of.Engines       87951 non-null  object
 8   Engine.Type             87951 non-null  object
 9   Purpose.of.flight       87951 non-null  object
 10  Weather.Condition       87951 non-null  object
 11  Broad.phase.of.flight   87951 non-null  object
 12  Report.Status           87951 non-null  object
 13  Total.Fatal.Injuries    87951 non-null  float64
 14  Total.Serious.Injuries  87951 non-null  float64
 15  Total.Minor.Injuries    87951 non-null  float64
 16  Total.Uninjured         87951 non-null  float64
 17  year                    87951 non-null  Int64
 18  Injury Severity         87951 non-null  object
 19  is_fatal                87951 non-null  int32
 20  is_severe               87951 non-null  int32
dtypes: Int64(1), datetime64[ns](1), float64(4), int32(2), object(13)
memory usage: 14.2+ MB
```

In [75]:

```
# check missing values across all six main analytical themes
theme_cols = [
    'Broad.phase.of.flight',
    'Weather.Condition',
    'Make',
    'Engine.Type',
    'Purpose.of.flight',
    'Number.of.Engines'
]

df_final[theme_cols].isna().sum()
```

Out[75]:

```
Broad.phase.of.flight    0
Weather.Condition        0
Make                     0
Engine.Type              0
Purpose.of.flight        0
Number.of.Engines        0
dtype: int64
```

Theme 1: Phase of flight vs. accident severity

This analysis examines how accident severity varies across different phases of flight - such as takeoff, climb, cruise, approach, and landing. The goal is to identify which phases are most prone to fatal or severe accidents. We will use the is_fatal and is_severe flags to calculate:

1. Total accidents per flight phase
2. Fatal rate (share of accidents with at least one death)
3. Severe rate (share of accidents with at least one death or serious injury)

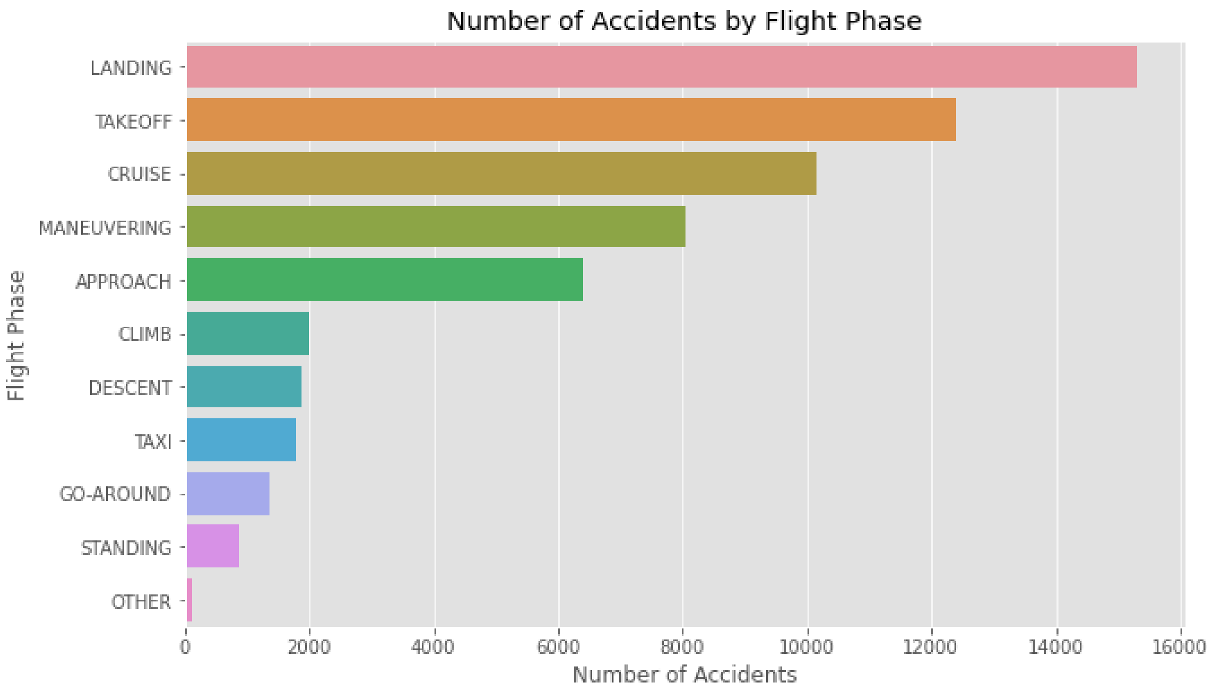
In [76]:
```
fight_phase_risk = (
    df_final.groupby('Broad.phase.of.flight', dropna=False)
    .agg(
        total_accidents=('is_fatal', 'size'),
        fatal_rate=('is_fatal', 'mean'),
        severe_rate=('is_severe', 'mean')
    )
    .sort_values('severe_rate', ascending=False)
)

fight_phase_risk
```

Out[76]:

Broad.phase.of.flight	total_accidents	fatal_rate	severe_rate
MANEUVERING	8052	0.387606	0.531918
OTHER	116	0.379310	0.456897
CLIMB	1995	0.299248	0.451629
DESCENT	1870	0.242246	0.395722
APPROACH	6389	0.240257	0.392080
UNKNOWN	27661	0.237555	0.387838
CRUISE	10141	0.267331	0.387043
STANDING	872	0.118119	0.349771
GO-AROUND	1345	0.196283	0.340520
TAKEOFF	12404	0.145679	0.288052
LANDING	15320	0.018473	0.070692
TAXI	1786	0.021277	0.064390

In [77]:
```
# --plot (accidents by flight phase ) excluding UNKNOWN or NaN phases
phase_counts = (
    df_final.loc[~df_final['Broad.phase.of.flight'].isin(['UNKNOWN', 'Unknown', 'NaN
    ['Broad.phase.of.flight']
    .value_counts()
)

# plot again
plt.figure(figsize=(10,6))
sns.barplot(y=phase_counts.index, x=phase_counts.values, orient='h')
plt.title('Number of Accidents by Flight Phase')
plt.xlabel('Number of Accidents')
plt.ylabel('Flight Phase')
```

```
plt.show()

## Results show that:
#1. Landing, cruise and takeoff have the highest number of accidents (about 40% of a
#2. Maneuvering and climb show fewer accidents overall, 9% and 2%, respectively), bu
#3. Descent and approach also have moderate levels of severity, but their accident f
#Operational safety measures should be enhanced for phases with high severity, even
#Safety improvements should target the Maneuvering and Climb phases, where the risk
```



Theme 2: weather conditions vs. accident severity

We will analyse accident risk by weather condition to:

1. To check if accidents are more likely to be fatal or severe in poor weather.
2. This identifies environmental risks that the aviation business must prepare for.

In [78]:

```
weather_risk = (
    df_final.groupby('Weather.Condition', dropna=False)
    .agg(
        total_accidents=('is_fatal', 'size'),
        fatal_rate=('is_fatal', 'mean'),
        severe_rate=('is_severe', 'mean')
    )
    .sort_values('severe_rate', ascending=False)
)

# convert rates to percentages for easier interpretation
weather_risk[['fatal_rate', 'severe_rate']] = weather_risk[['fatal_rate', 'severe_ra
weather_risk = weather_risk.round({'fatal_rate': 2, 'severe_rate': 2})

weather_risk

#Results explanation
# 1. IMC (bad weather conditions): The highest fatal rate (58.13%) and severe rate (
# 2. VMC (clear weather): Fatal rate of 15.68% and severe rate of 28.95%, showing th
# 3. The aviation business should pay special attention to IMC conditions, where the
```

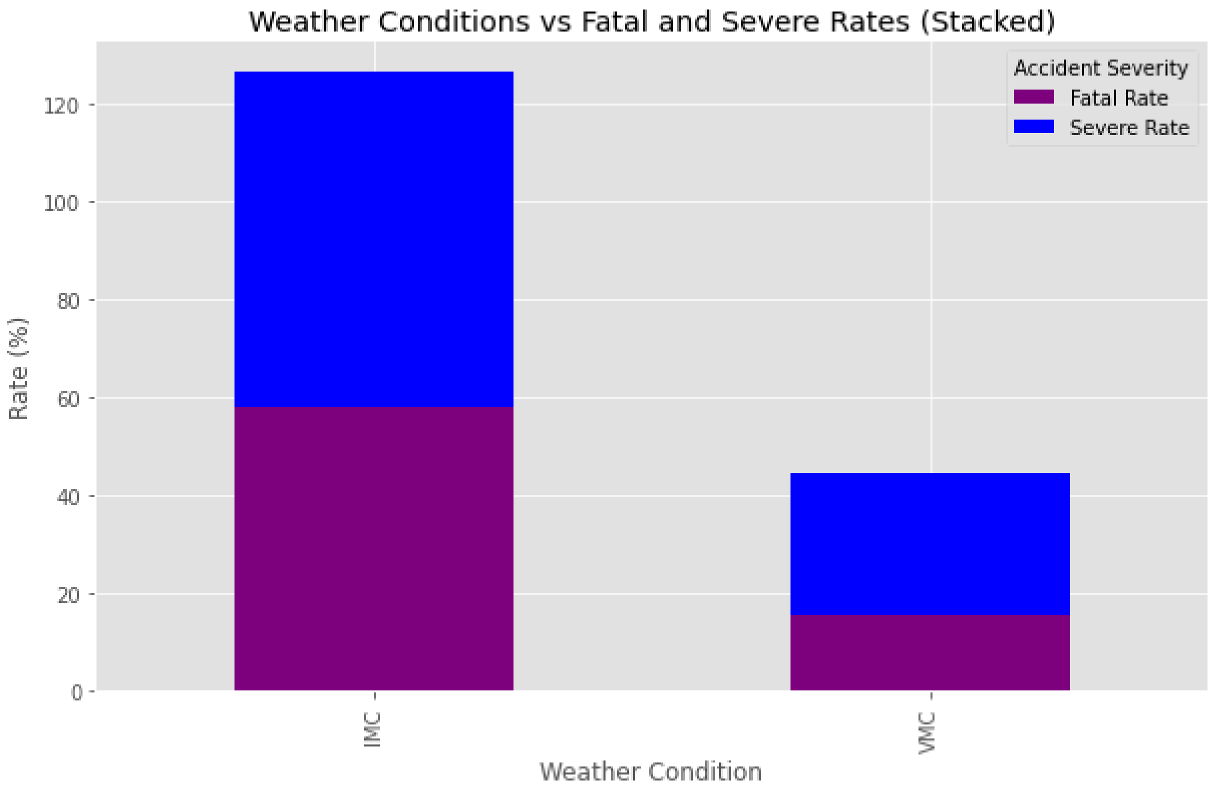
Out[78]:

	total_accidents	fatal_rate	severe_rate
Weather.Condition			
IMC	5949	58.13	68.65
UNKNOWN	5585	37.33	44.10
VMC	76417	15.68	28.95

In [79]:

```
# --plot (accidents by weather conditions ) -excluding "UNKNOWN"
weather_risk_clean = weather_risk.drop(index='UNKNOWN')

# Plot the results
plt.figure(figsize=(10,6))
weather_risk_clean[['fatal_rate', 'severe_rate']].plot(kind='bar', stacked=True, col

plt.title('Weather Conditions vs Fatal and Severe Rates (Stacked)')
plt.xlabel('Weather Condition')
plt.ylabel('Rate (%)')
plt.legend(title='Accident Severity', labels=['Fatal Rate', 'Severe Rate'])
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```



Theme 3: analyze accident severity by aircraft make and model

We will analyse accident risk by aircraft make and model to:

1. To check which aircraft types have higher or lower accident severity,
2. T0 guiding safer investment decisions in aircrafts

In [80]:

```
df_final.groupby('Make')[['Total.Fatal.Injuries', 'Total.Serious.Injuries']].agg(['s
```

Out[80]:

	Total.Fatal.Injuries		Total.Serious.Injuries	
	sum	mean	sum	mean
Make				
107.5 FLYING CORPORATION	1.0	1.0	0.0	0.0
1200	0.0	0.0	1.0	1.0
177MF LLC	0.0	0.0	2.0	2.0
1977 COLFER-CHAN	0.0	0.0	0.0	0.0
1ST FTR GP	1.0	1.0	0.0	0.0
...	...	...	...	...
ZUBAIR S KHAN	1.0	1.0	0.0	0.0
ZUBER THOMAS P	0.0	0.0	0.0	0.0
ZUKOWSKI	0.0	0.0	0.0	0.0
ZWART	0.0	0.0	0.0	0.0
ZWICKER MURRAY R	0.0	0.0	0.0	0.0

7551 rows × 4 columns

In [81]:

```
# Summarize total accidents, mean fatal rate, and mean severe rate per manufacturer
make_summary = (
    df_final.groupby('Make')
    .agg(
        total_accidents=('Event.Id', 'count'),
        avg_fatal_rate=('Total.Fatal.Injuries', 'mean'),
        avg_severe_rate=('Total.Serious.Injuries', 'mean')
    )
    .sort_values(by='total_accidents', ascending=False)
)

# Display top 10 manufacturers
make_summary.head(10)
```

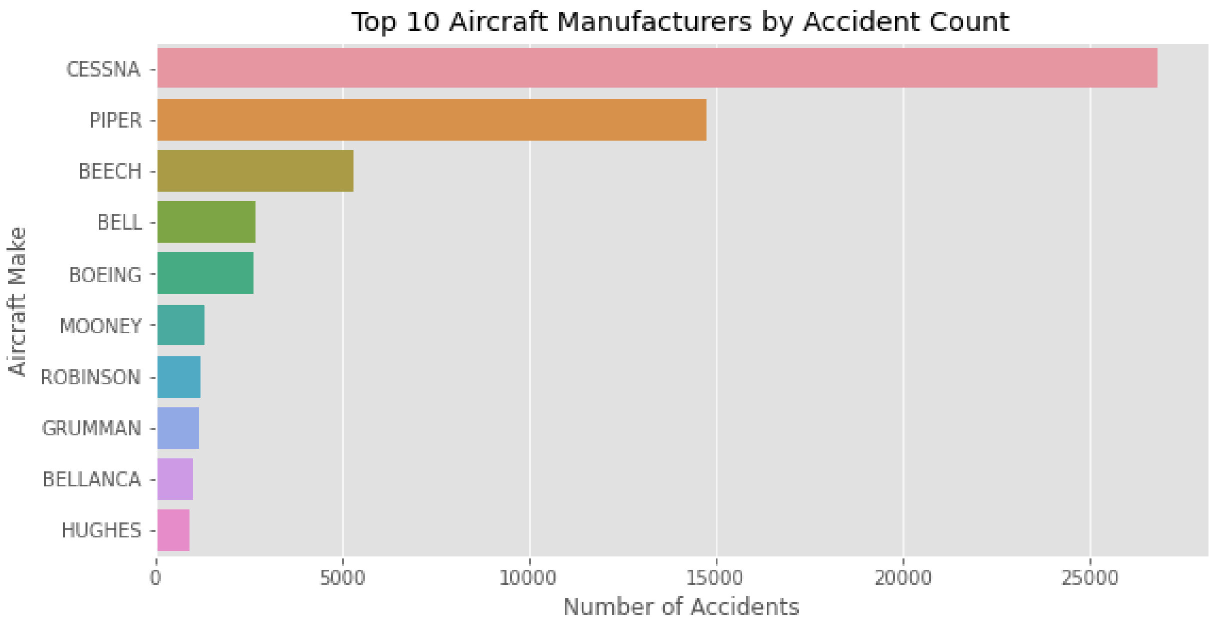
Out[81]:

	total_accidents	avg_fatal_rate	avg_severe_rate
Make			
CESSNA	26839	0.348076	0.180707
PIPER	14744	0.444995	0.206389
BEECH	5332	0.694299	0.204989
BELL	2706	0.484848	0.320769
BOEING	2652	3.059578	0.801659
MOONEY	1322	0.512859	0.187595
ROBINSON	1223	0.497138	0.181521
GRUMMAN	1158	0.202936	0.139033
BELLANCA	1033	0.331075	0.189739

Make	total_accidents	avg_fatal_rate	avg_severe_rate
HUGHES	931	0.218045	0.245972

In [82]:

```
# top 10 aircraft makes
make_counts = df_final['Make'].value_counts().head(10)

plt.figure(figsize=(10,5))
sns.barplot(x=make_counts.values, y=make_counts.index)
plt.title('Top 10 Aircraft Manufacturers by Accident Count')
plt.xlabel('Number of Accidents')
plt.ylabel('Aircraft Make')
plt.show()
```



Theme 4: analyze accident severity by purpose of flight

We will analyse accident risk by aircraft make and model to:

1. to help identify which flight operations (personal, business, training, etc.) carry higher or lower risk
2. to help guide what kind of aviation activities are safer for a new business to pursue.

In [83]:

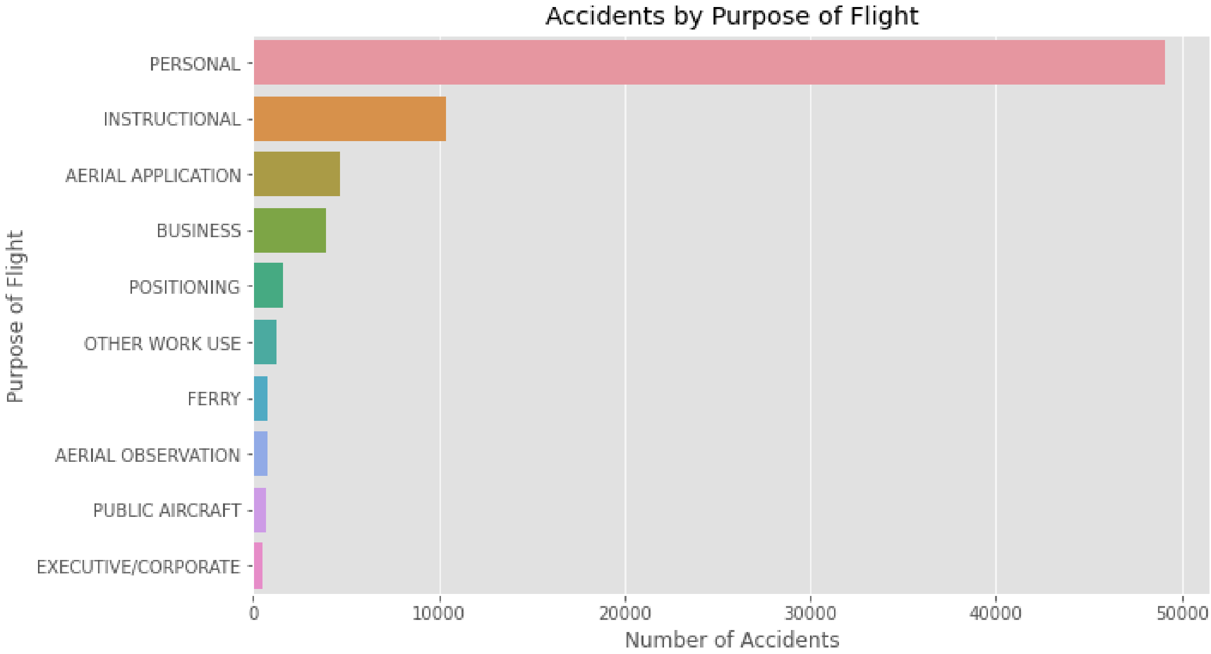
```
purpose_risk = (
    df_final.groupby('Purpose.of.flight', dropna=False)
    .agg(
        total_accidents=('is_fatal', 'size'),
        fatal_rate=('is_fatal', 'mean'),
        severe_rate=('is_severe', 'mean')
    )
    .sort_values('severe_rate', ascending=False)
)

# convert to percentages for easier reading
purpose_risk[['fatal_rate', 'severe_rate']] = purpose_risk[['fatal_rate', 'severe_ra
purpose_risk = purpose_risk.round({'fatal_rate': 2, 'severe_rate': 2})

purpose_risk
```

Out[83]:

Purpose.of.flight	total_accidents	fatal_rate	severe_rate
ASHO	5	60.00	80.00
AIR RACE/SHOW	53	45.28	77.36
FIREFIGHTING	40	47.50	60.00
AIR RACE SHOW	99	39.39	52.53
SKYDIVING	181	32.04	49.17
GLIDER TOW	53	26.42	47.17
AERIAL OBSERVATION	787	28.08	47.01
AIR DROP	11	27.27	45.45
BANNER TOW	101	15.84	43.56
PUBLIC AIRCRAFT - STATE	64	18.75	40.62
PUBLIC AIRCRAFT - LOCAL	74	12.16	40.54
FLIGHT TEST	405	20.49	38.27
EXTERNAL LOAD	123	24.39	38.21
OTHER WORK USE	1250	20.40	38.16
BUSINESS	3971	26.42	37.17
PUBLIC AIRCRAFT	710	23.94	37.04
PUBLIC AIRCRAFT - FEDERAL	104	19.23	36.54
UNKNOWN	12731	24.15	35.68
PERSONAL	49076	20.94	34.55
EXECUTIVE/CORPORATE	542	24.91	34.50
POSITIONING	1632	23.71	32.97
FERRY	806	20.72	30.40
AERIAL APPLICATION	4686	10.73	22.90
INSTRUCTIONAL	10442	9.18	18.78
PUBL	1	0.00	0.00
PUBS	4	0.00	0.00

In [84]:

```
# -- plot (Accidents by purpose of flight)
df_cleaned = df_final[df_final['Purpose.of.flight'] != 'UNKNOWN']     # Remove 'UNKNO

purpose_counts = df_cleaned['Purpose.of.flight'].value_counts().head(10)

plt.figure(figsize=(10,6))
sns.barplot(y=purpose_counts.index, x=purpose_counts.values, orient='h')
plt.title('Accidents by Purpose of Flight')
plt.xlabel('Number of Accidents')
plt.ylabel('Purpose of Flight')
plt.show()

#High accident frequency does not necessarily correlate with high severity. PERSONAL
#High-risk operations like ASHO and FIREFIGHTING may have fewer accidents but much h
```



Theme 5: analyze accident severity by engine type

We will analyse accident risk by aircraft engine type to:

1. to help identify whether engine type often reflects the design and performance class of an aircraft.
2. to help understand the severity by engine type helps guide safe investment decisions in technology.

In [85]:

```
engine_risk = (
    df_final.groupby('Engine.Type', dropna=False)
    .agg(
        total_accidents=('is_fatal', 'size'),
        fatal_rate=('is_fatal', 'mean'),
        severe_rate=('is_severe', 'mean')
    )
    .sort_values('severe_rate', ascending=False)
)

# convert to percentages for readability
engine_risk[['fatal_rate', 'severe_rate']] = engine_risk[['fatal_rate', 'severe_rate
engine_risk = engine_risk.round({'fatal_rate': 2, 'severe_rate': 2})

engine_risk
```

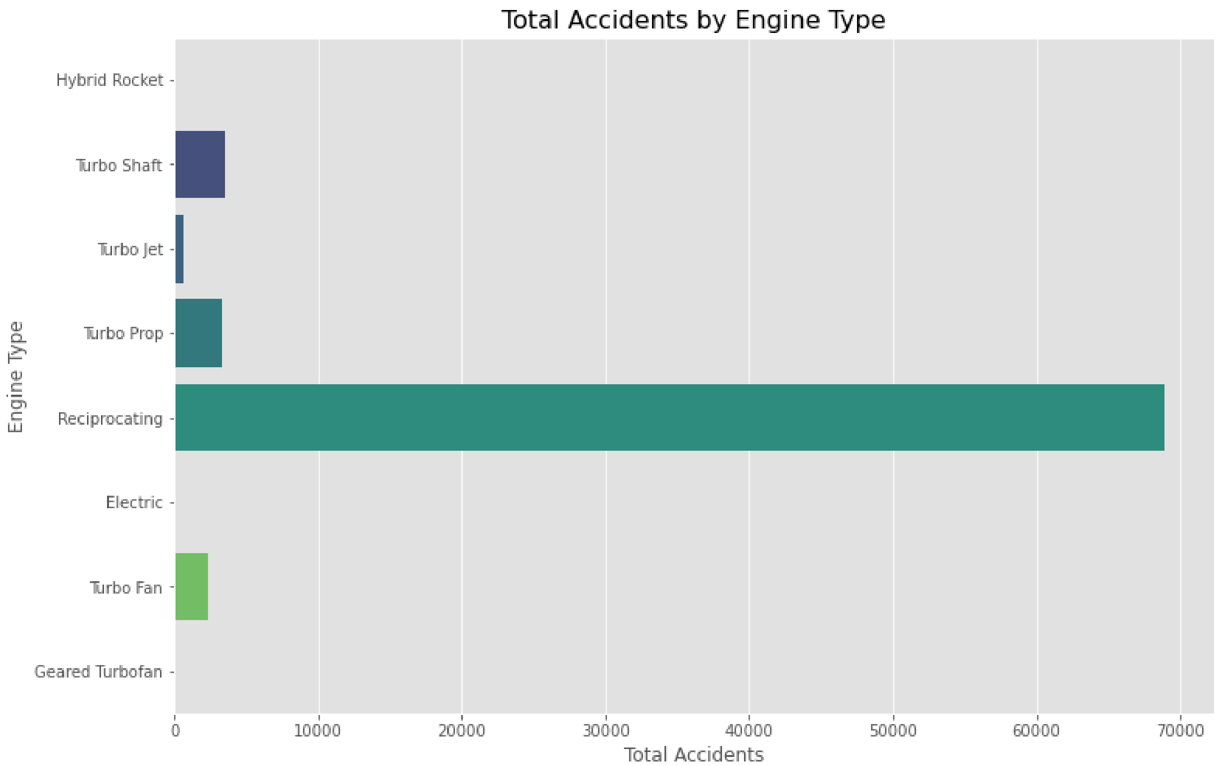
Out[85]:

Engine.Type	total_accidents	fatal_rate	severe_rate
Hybrid Rocket	1	100.00	100.00
Unknown	9065	29.62	44.36
Turbo Shaft	3583	21.88	37.76
Turbo Jet	684	19.15	34.50
Turbo Prop	3324	23.80	33.63
Reciprocating	68885	18.79	30.96

	total_accidents	fatal_rate	severe_rate
Engine.Type			
Electric	10	20.00	30.00
Turbo Fan	2387	7.88	25.43
Geared Turbofan	12	0.00	0.00

In [86]:

```
# -- plot (Accidents by engine type)

engine_risk = engine_risk.reset_index()                # Resetting the index ensure

engine_risk['Engine.Type'] = engine_risk['Engine.Type'].astype(str)
engine_risk_clean = engine_risk[~engine_risk['Engine.Type'].str.contains('unknown',

plt.figure(figsize=(12,8))     # Plot total accidents by engine type (excluding 'Unk
sns.barplot(x='total_accidents', y='Engine.Type', data=engine_risk_clean, palette='v
plt.title('Total Accidents by Engine Type', fontsize=16)
plt.xlabel('Total Accidents', fontsize=12)
plt.ylabel('Engine Type', fontsize=12)
plt.show()

#Results
# 1. High-frequency, low-severity engines: Reciprocating engines, though involved in
# 2. High-risk engines: Turbo Shaft and Turbo Prop engines, although less frequent, 
#3. High-risk engines: Turbo Shaft and Turbo Prop engines, although less frequent, h
# For fleet expansion: Prioritize engines like Reciprocating (high frequency, lower 
# For improving safety protocols: Focus on Turbo Shaft and Turbo Prop engines, as th
```



In [87]:

```
print(engine_risk.columns)
```

```
Index(['Engine.Type', 'total_accidents', 'fatal_rate', 'severe_rate'], dtype='objec
t')
```

Theme 6: analyze accident severity by number of engines

We will analyse accident risk by aircraft engine type to:

1. to help identify whether number of engines often reflects the design and performance class of an aircraft.
2. to help understand the severity by number of engines helps guide safe investment decisions in technology.

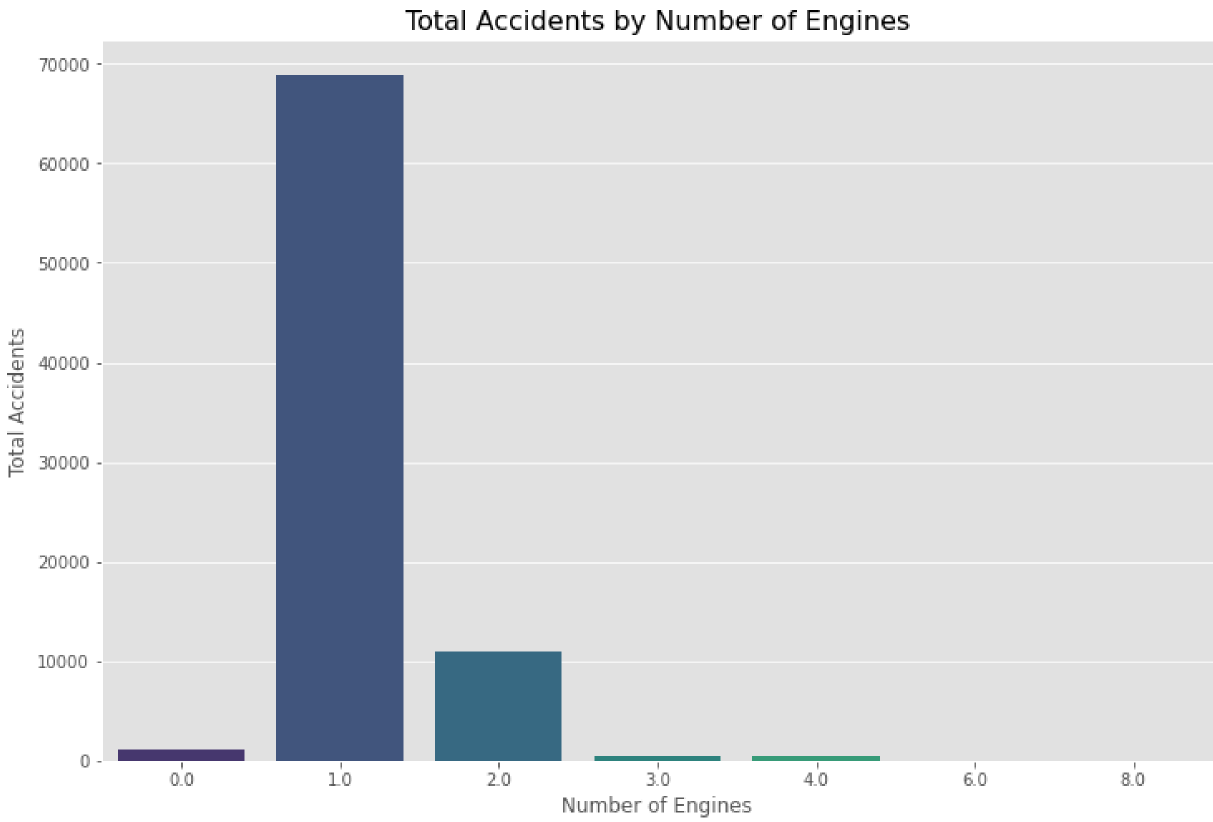
In [88]:
```
# analyze severity by number of engines
number_of_engines_risk = (
    df_final.groupby('Number.of.Engines', dropna=False)
    .agg(
        total_accidents=('is_fatal', 'size'),
        fatal_rate=('is_fatal', 'mean'),
        severe_rate=('is_severe', 'mean')
    )
    .sort_values('severe_rate', ascending=False)
)

# convert to percentages for easier interpretation
number_of_engines_risk[['fatal_rate', 'severe_rate']] = (
    number_of_engines_risk[['fatal_rate', 'severe_rate']] * 100
)
number_of_engines_risk = number_of_engines_risk.round({'fatal_rate': 2, 'severe_rate

number_of_engines_risk
```

Out[88]:

Number.of.Engines	total_accidents	fatal_rate	severe_rate
8.0	3	0.00	66.67
Unknown	6027	33.73	46.01
0.0	1210	13.31	39.92
2.0	10891	24.85	35.63
1.0	68956	18.21	30.93
4.0	415	12.77	24.58
3.0	448	4.24	22.32
6.0	1	0.00	0.00

In [89]:
```
# -- plot (Accidents by engine type)
number_of_engines_risk_clean = number_of_engines_risk.reset_index()

number_of_engines_risk_clean = number_of_engines_risk_clean[number_of_engines_risk_c
plt.figure(figsize=(12,8))
sns.barplot(x='Number.of.Engines', y='total_accidents', data=number_of_engines_risk_
plt.title('Total Accidents by Number of Engines', fontsize=16)
plt.xlabel('Number of Engines', fontsize=12)
plt.ylabel('Total Accidents', fontsize=12)
plt.show()

#Results
# 1. Aircraft with 1 engine have the highest accident frequency (68,956 accidents),
# 2-engine aircraft show higher severity, with 10,891 accidents, a fatal rate of 24.
# 3-engine aircraft have fewer accidents (448) but lower severity (fatal rate 4.24%,
# Aircraft with 4 engines show moderate severity (fatal rate 12.77%, severe rate 24.
# while 6-engine aircraft have an outlier with only 1 accident.
```

```
#Focus on improving safety protocols for 2-engine aircraft, as they show both modera
#Continue monitoring 1-engine aircraft for safety, given their high accident frequen
```



Step 8: Synthesis of the top three key themes with strongest recommendations.

After analyzing the dataset above, we have identified the top three themes that are most critical for a business diversifying into aviation. These themes directly impact safety, risk management, and operational efficiency, all crucial factors for the business.

1. Flight Phases

-The Maneuvering and Climb flight phases, although less frequent, exhibit the highest severity with fatal rates as high as 38.8%. Despite these phases contributing to a smaller percentage of total accidents, the fatal and severe accident rates are significant, indicating a need for heightened focus on risk mitigation in these phases.

Recommendation: Implement advanced safety protocols tailored to high-risk flight phases such as Maneuvering and Climb. This could include specialized training for pilots, better safety monitoring systems, and automated flight controls to reduce human error in these phases. Although their frequency is lower, their high impact on fatal accidents justifies the focus.

2. Weather Conditions (IMC)

-IMC (Instrument Meteorological Conditions) conditions have a significantly higher fatal and severe accident rate, with fatal rates reaching 58.13% and severe rates reaching 68.65%. Poor weather conditions such as fog, heavy rain, and thunderstorms greatly affect safety.

Recommendation: Invest in weather-adaptive aircraft and pilot training specifically focused on handling IMC conditions. Additionally, implementing real-time weather tracking and advanced navigation systems can significantly reduce the severity of accidents in adverse weather conditions. This focus on weather safety ensures fewer weather-related accidents and increased operational continuity, which is critical for business reliability and customer trust.

3. Aircraft Make and Model

-Aircraft make and model have been shown to contribute significantly to both the total number of accidents and the severity of those accidents. Some aircraft types exhibit lower accident severity while others have higher fatality rates despite fewer accidents. Choosing the right aircraft models with proven safety records is crucial for minimizing risk.

Recommendation: When expanding the fleet, the business should prioritize aircraft with a strong safety record and lower accident severity. A deeper dive into the historical accident data of different aircraft makes and models will help guide fleet investment decisions, ensuring that they are buying into aircraft that will have lower long-term risk and maintenance costs. This not only enhances safety but also helps in managing operational costs and insurance premium

In [45]:
```
pip install openpyxl
```

```
Requirement already satisfied: openpyxl in c:\users\abigaelkariuki\anaconda3\envs\lea
rn-env\lib\site-packages (3.0.5)
Requirement already satisfied: jdcal in c:\users\abigaelkariuki\anaconda3\envs\learn-
env\lib\site-packages (from openpyxl) (1.4.1)
Requirement already satisfied: et-xmlfile in c:\users\abigaelkariuki\anaconda3\envs\l
earn-env\lib\site-packages (from openpyxl) (1.0.1)
Note: you may need to restart the kernel to use updated packages.
```

In [46]:
```
import re

# Define regex for illegal Excel characters
ILLEGAL_CHARACTERS_RE = re.compile(r'[\000-\010]|[\013-\014]|[\016-\037]')

# Clean all string cells in df_final
df_final = df_final.applymap(lambda x: ILLEGAL_CHARACTERS_RE.sub('', x) if isinstanc
```

In [47]:
```
df_final.to_excel(r"C:\Users\AbigaelKariuki\Documents\cleaned_aviation_data.xlsx", i
```