Aviation Accident Database & Synopses, up to 2023

Introduction.

Business understanding

A company is embarking on a strategic expansion into the aviation sector, aiming to diversify its investment portfolio by purchasing and operating planes for both commercial and private enterprises. As this venture is new to the organisation, there is limited internal knowledge about the aviation domain, particularly regarding safety and operational risk.

Business problem

The company must identify which aircraft models and types have the lowest operational risk to guide safe and cost-effective entry into the aviation market.

EDA objectives

This notebook explores aircraft risk patterns to support the Company's on safe aircraft acquisition decisions. The following objectives are structured around key EDA phases:

1. **Data understanding**
    - Load and preview the dataset to understand its structure, size and key variables.
    - Identify the most relevant columns for analyzing aviation safety.
2. **Data cleaning**
    - Handle missing values and duplicated values.
    - Check on inconsistent formatting in the key fields.
    - Standardize categorical variables for accurate grouping.
    - Check on outliers.......
3. **Univariate & Bivariate Analysis**
    - Explore the most common aircraft types and count how many incidents each has.
    - Analyze incident trends over time to detect rising or declining risk patterns
4. **Multivariate & Grouped Insights**
    - Compare incident frequencies by usage type (commercial vs private), age, or manufacturer.
5. **Incident Causes & Locations**
    - Visualize the top causes of incidents and the regions with the highest incident density.

1. Data Understanding

- **Import the necessary libraries required for the data.**
- **Load the `Aviation.csv` data**

In [1]:
```
# import the necessary libraries.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
# Load the data.
df= pd.read_csv('AviationData.csv', encoding = 'latin1', low_memory = False)
```

In [5]:
```
# Check the size of the data
df.shape
```

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

In [7]:
```
# check the columns of the data
df.columns
```

Out[7]:
```
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 [9]:
```
# check the whole iformation  of the data
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            50132 non-null  object 
 9   Airport.Name            52704 non-null  object 
 10  Injury.Severity         87889 non-null  object 
 11  Aircraft.damage         85695 non-null  object 
 12  Aircraft.Category       32287 non-null  object 
 13  Registration.Number     87507 non-null  object 
 14  Make                    88826 non-null  object 
 15  Model                   88797 non-null  object 
 16  Amateur.Built           88787 non-null  object 
 17  Number.of.Engines       82805 non-null  float64
 18  Engine.Type             81793 non-null  object 
 19  FAR.Description         32023 non-null  object 
 20  Schedule                12582 non-null  object 
 21  Purpose.of.flight       82697 non-null  object 
 22  Air.carrier             16648 non-null  object 
 23  Total.Fatal.Injuries    77488 non-null  float64
 24  Total.Serious.Injuries  76379 non-null  float64
 25  Total.Minor.Injuries    76956 non-null  float64
 26  Total.Uninjured         82977 non-null  float64
 27  Weather.Condition       84397 non-null  object 
 28  Broad.phase.of.flight   61724 non-null  object 
 29  Report.Status           82505 non-null  object 
 30  Publication.Date        75118 non-null  object 
dtypes: float64(5), object(26)
memory usage: 21.0+ MB
```

In [11]:

```
# check the first five rows
df.head()
```

Out[11]:

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

5 rows × 31 columns

In [13]:

```
# check the last 5 rows
df.tail()
```

Out[13]:

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

5 rows × 31 columns

In [15]:

```
# check the statistical summary of the data
df.describe()
```

Out[15]:

	Number.of.Engines	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries	Tot
count	82805.000000	77488.000000	76379.000000	76956.000000	8
mean	1.146585	0.647855	0.279881	0.357061	
std	0.446510	5.485960	1.544084	2.235625	
min	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	0.000000	0.000000	0.000000	
50%	1.000000	0.000000	0.000000	0.000000	
75%	1.000000	0.000000	0.000000	0.000000	
max	8.000000	349.000000	161.000000	380.000000	

For the column to be relevant to the analysis, it should answer the questions:

- Which? Answer types of aircraft or models.
- Why? Answers what caused the incident
- When? Answer the time trend
- What kind of incident happened? Answer types of incidents and which one of them is sever

2. Data cleaning

- Handle missing values and duplicated values.
- Check on inconsistent formatting in the key fields.
- Standardize categorical variables for accurate grouping.
- Check on outliers.......

In [17]:

```
# check on duplicated values
df.duplicated().any()
```

Out[17]: False

In [19]:

```
# Check out the columns with  missing values
df.isnull().any()
```

Out[19]:
```
Event.Id                  False
Investigation.Type        False
Accident.Number           False
Event.Date                False
Location                   True
Country                    True
Latitude                   True
Longitude                  True
Airport.Code               True
Airport.Name               True
Injury.Severity            True
Aircraft.damage            True
Aircraft.Category          True
Registration.Number        True
Make                       True
Model                      True
Amateur.Built              True
Number.of.Engines          True
Engine.Type                True
FAR.Description            True
Schedule                   True
Purpose.of.flight          True
Air.carrier                True
Total.Fatal.Injuries       True
Total.Serious.Injuries     True
Total.Minor.Injuries       True
Total.Uninjured            True
Weather.Condition          True
Broad.phase.of.flight      True
Report.Status              True
Publication.Date           True
dtype: bool
```

In [21]:
```
# check the columns missing values with their percentages
df.isnull().mean().sort_values(ascending=False).round(2) * 100
```

Out[21]:
```
Schedule                   86.0
Air.carrier                81.0
FAR.Description            64.0
Aircraft.Category          64.0
Longitude                  61.0
Latitude                   61.0
Airport.Code               44.0
Airport.Name               41.0
Broad.phase.of.flight      31.0
Publication.Date           15.0
Total.Serious.Injuries     14.0
Total.Minor.Injuries       13.0
Total.Fatal.Injuries       13.0
Engine.Type                 8.0
Report.Status               7.0
Purpose.of.flight           7.0
Number.of.Engines           7.0
Total.Uninjured             7.0
Weather.Condition           5.0
Aircraft.damage             4.0
Registration.Number         2.0
Injury.Severity             1.0
Country                     0.0
Amateur.Built               0.0
Model                       0.0
Make                        0.0
Location                    0.0
Investigation.Type          0.0
Event.Date                  0.0
Accident.Number             0.0
Event.Id                    0.0
dtype: float64
```

In [23]:
```
#  Drop the unnecessary  columns, I mean according to the analysis
df.drop(['Schedule','Airport.Code', 'Registration.Number'],axis=1, inplace= True)
```

In [25]:
```
# Replace columns with unknown values.
# This is to avoid  biased data.
df['Air.carrier'] = df['Air.carrier'].fillna('Unknown Operator')
df['FAR.Description'] = df['FAR.Description'].fillna('Unknown FAR Category')
df['Aircraft.Category'] = df['Aircraft.Category'].fillna('Unknown Category')
df['Airport.Name']= df['Airport.Name'].fillna('Unknown Airport')
df['Broad.phase.of.flight']=df['Broad.phase.of.flight'].fillna('Unknown Phase')
```

In [27]:
```
#  check for the remaining columns with missing values .
df.isnull().sum().sort_values(ascending=False)
```

Out[27]:

```
Longitude                 54516
Latitude                  54507
Publication.Date          13771
Total.Serious.Injuries    12510
Total.Minor.Injuries      11933
Total.Fatal.Injuries      11401
Engine.Type                7096
Report.Status              6384
Purpose.of.flight          6192
Number.of.Engines          6084
Total.Uninjured            5912
Weather.Condition          4492
Aircraft.damage            3194
Injury.Severity            1000
Country                     226
Amateur.Built               102
Model                        92
Make                         63
Location                     52
Investigation.Type            0
FAR.Description               0
Air.carrier                   0
Aircraft.Category             0
Airport.Name                  0
Event.Date                    0
Broad.phase.of.flight         0
Accident.Number               0
Event.Id                      0
dtype: int64
```

In [29]:

```python
#   lets check the mean and median of this columns
cols_int = [
    'Total.Serious.Injuries',
    'Total.Minor.Injuries',
    'Total.Fatal.Injuries',
    'Total.Uninjured',
    'Number.of.Engines'
]

# Check mean for all at once
df[cols_int].mean()

# check medians for the columns
df[cols_int].median()
```

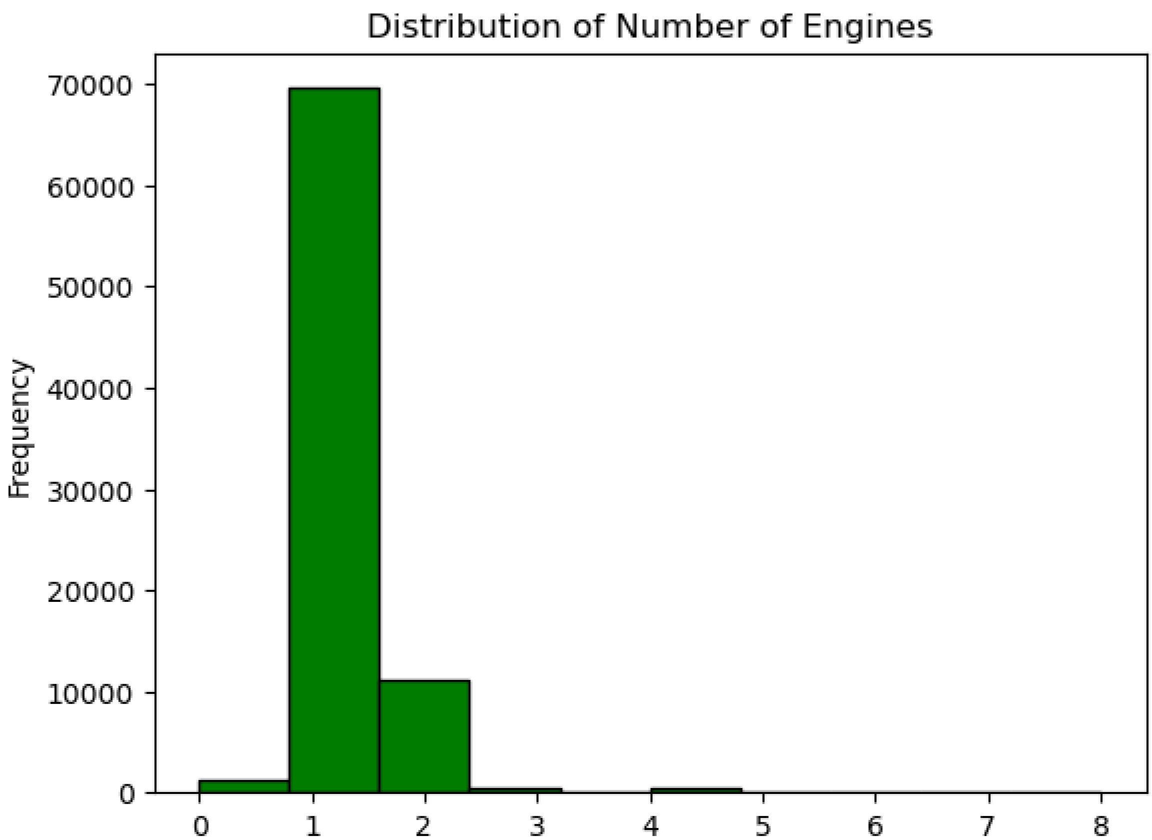
Out[29]:

```
Total.Serious.Injuries    0.0
Total.Minor.Injuries      0.0
Total.Fatal.Injuries      0.0
Total.Uninjured           1.0
Number.of.Engines         1.0
dtype: float64
```

So, according to the data above we cannot fill the missing values of the injured columns with the mean and median but we can fill them with `0`

In [31]:
```
# fill the missing values of the injured cols with 0
injury_cols = ['Total.Serious.Injuries', 'Total.Minor.Injuries', 'Total.Fatal.Injur
df[injury_cols] = df[injury_cols].fillna(0)
```

In [33]:
```
#  lets look into the number.of. Engines col using a visual to checkif the data is
import matplotlib.pyplot as plt

df['Number.of.Engines'].dropna().plot(kind='hist', bins=10, color='green', ec= 'bla
plt.title("Distribution of Number of Engines")
plt.show()
```



In [35]:
```
# fill the data  with median
df['Number.of.Engines']= df['Number.of.Engines'].fillna(df['Number.of.Engines'].med
```

In [37]:
```
#  check for the remaining columns with missing values .
df.isnull().sum().sort_values(ascending=False)
```

Out[37]:
```
Longitude                 54516
Latitude                  54507
Publication.Date          13771
Engine.Type                7096
Report.Status              6384
Purpose.of.flight          6192
Weather.Condition          4492
Aircraft.damage            3194
Injury.Severity            1000
Country                     226
Amateur.Built               102
Model                        92
Make                         63
Location                     52
Aircraft.Category             0
Total.Serious.Injuries        0
Accident.Number               0
Broad.phase.of.flight         0
Event.Date                    0
Total.Uninjured               0
Total.Minor.Injuries          0
Air.carrier                   0
Total.Fatal.Injuries          0
FAR.Description               0
Airport.Name                  0
Number.of.Engines             0
Investigation.Type            0
Event.Id                      0
dtype: int64
```

In [39]:
```
# ckeck the injury.severity col
df[df['Injury.Severity'].isnull()]
```

Out[39]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Coun
63918	20080111X00038	Incident	DCA08WA024	2008-01-03	Deauville Saint, France	Fra
63962	20080204X00132	Accident	NYC08WA081	2008-01-16	Kiteni, Peru	P
63987	20080304X00254	Incident	ENG08RA015	2008-01-24	Kingston, Jamaica	Jama
64026	20081219X65255	Incident	ENG08WA014	2008-02-03	Nurnberg, Germany	Germ
64128	20080409X00444	Accident	NYC08WA121	2008-02-28	Lago Ranco, Chile	Cl
...	...	...	...	...	...	
88863	20221213106449	Accident	GAA22WA311	2022-12-11	Kildare,	Irela
88874	20221215106462	Accident	CEN23LA064	2022-12-15	Patterson, LA	Uni Sta
88879	20221219106472	Accident	DCA23LA096	2022-12-18	Kahului, HI	Uni Sta
88885	20221227106494	Accident	ERA23LA095	2022-12-26	Hampton, NH	Uni Sta
88887	20221227106498	Accident	WPR23LA076	2022-12-26	Morgan, UT	Uni Sta

1000 rows × 28 columns

In [41]:
```
# We can fill up this column with the injury columns .

def refer_severity(row):
    if row['Total.Fatal.Injuries'] > 0:
        return 'Fatal'
    elif row['Total.Serious.Injuries'] > 0:
        return 'Serious'
    elif row['Total.Minor.Injuries'] > 0:
        return 'Minor'
    elif row['Total.Uninjured'] > 0:
        return 'None'
    else:
        return 'Unknown'

df['Injury.Severity'] = df.apply(refer_severity, axis=1)
```

In [43]:
```
df['Injury.Severity'].value_counts()
```

Out[43]:
```
Injury.Severity
None       47089
Fatal      17813
Minor      11488
Serious    11190
Unknown     1309
Name: count, dtype: int64
```

In [45]:
```
# check the  country and location
df[df['Country'].isnull()]
```

Out[45]:

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location
36	20020917X02410	Accident	MIA82FKA05	1982-01-04	SAINT CROIX
464	20020917X02358	Accident	MIA82DA062	1982-03-02	HUMA CAO
465	20020917X02026	Accident	FTW82DA076	1982-03-02	MUSTANG BLK A11
725	20020917X02377	Accident	MIA82DA091	1982-03-31	MOCA
831	20020917X02069	Accident	FTW82DA127	1982-04-13	WEST DELTA 105D
...	...	...	...	...	...
52288	20020322X00387	Accident	DCA02MA029	2002-03-22	int'l waters
54284	20040528X00699	Accident	DCA03WA031	2003-03-12	JOHANNESBURG
56200	20040528X00697	Accident	DCA04WA026	2004-02-10	Sharjah Airport
58803	20050616X00790	Incident	DCA05WA073	2005-06-10	Canada/US borde
62530	20070518X00582	Incident	DCA07WA043	2007-05-13	London Control

226 rows × 28 columns

In [47]:

```
# check on the  remaining cols with missing values.
df.isnull().sum().sort_values(ascending=False)
```

Out[47]:
```
Longitude                 54516
Latitude                  54507
Publication.Date          13771
Engine.Type                7096
Report.Status              6384
Purpose.of.flight          6192
Weather.Condition          4492
Aircraft.damage            3194
Country                     226
Amateur.Built               102
Model                        92
Make                         63
Location                     52
Injury.Severity               0
Total.Serious.Injuries        0
Accident.Number               0
Broad.phase.of.flight         0
Event.Date                    0
Total.Uninjured               0
Total.Minor.Injuries          0
Air.carrier                   0
Total.Fatal.Injuries          0
Aircraft.Category             0
FAR.Description               0
Airport.Name                  0
Number.of.Engines             0
Investigation.Type            0
Event.Id                      0
dtype: int64
```

In [49]:
```python
#Fill the remaining columns with unknowns.
# To prevent data irrelevance or bias.
rem_cols=['Engine.Type','Report.Status','Purpose.of.flight','Weather.Condition','Ai
          'Country','Amateur.Built','Model','Make','Location']
df[rem_cols]= df[rem_cols].fillna('Unknown')
```

In [51]:
```python
# check on the  remaining cols with missing values.
df.isnull().sum().sort_values(ascending=False)
```

Out[51]:
```
Longitude                  54516
Latitude                   54507
Publication.Date           13771
Number.of.Engines              0
Report.Status                  0
Broad.phase.of.flight          0
Weather.Condition              0
Total.Uninjured                0
Total.Minor.Injuries           0
Total.Serious.Injuries         0
Total.Fatal.Injuries           0
Air.carrier                    0
Purpose.of.flight              0
FAR.Description                0
Engine.Type                    0
Event.Id                       0
Investigation.Type             0
Model                          0
Make                           0
Aircraft.Category              0
Aircraft.damage                0
Injury.Severity                0
Airport.Name                   0
Country                        0
Location                       0
Event.Date                     0
Accident.Number                0
Amateur.Built                  0
dtype: int64
```

In [53]:
```
#  lets clean the latitude and longitude columns for Tableau mappings
#Ensure 'Latitude' and 'Longitude' are numeric
df['Latitude'] = pd.to_numeric(df['Latitude'], errors='coerce')
df['Longitude'] = pd.to_numeric(df['Longitude'], errors='coerce')
```

In [55]:
```
# Drop rows without coordinates needed for mapping
df = df.dropna(subset=['Latitude', 'Longitude'])
```

In [57]:
```
# Start fresh with a clean copy if you filtered before
df = df.copy()

#apply your parsing safely
df['Publication.Date'] = pd.to_datetime(df['Publication.Date'], format='%d-%m-%Y',
```

In [59]:
```
#create a display version that fills missing with 'Unknown'
df['PubDate_Display'] = df['Publication.Date'].dt.strftime('%Y-%m-%d')
df['PubDate_Display'] = df['PubDate_Display'].fillna('Unknown')
```

In [61]:
```
# counter check if the data is ready for use.
df.head(15)
```

Out[61]:

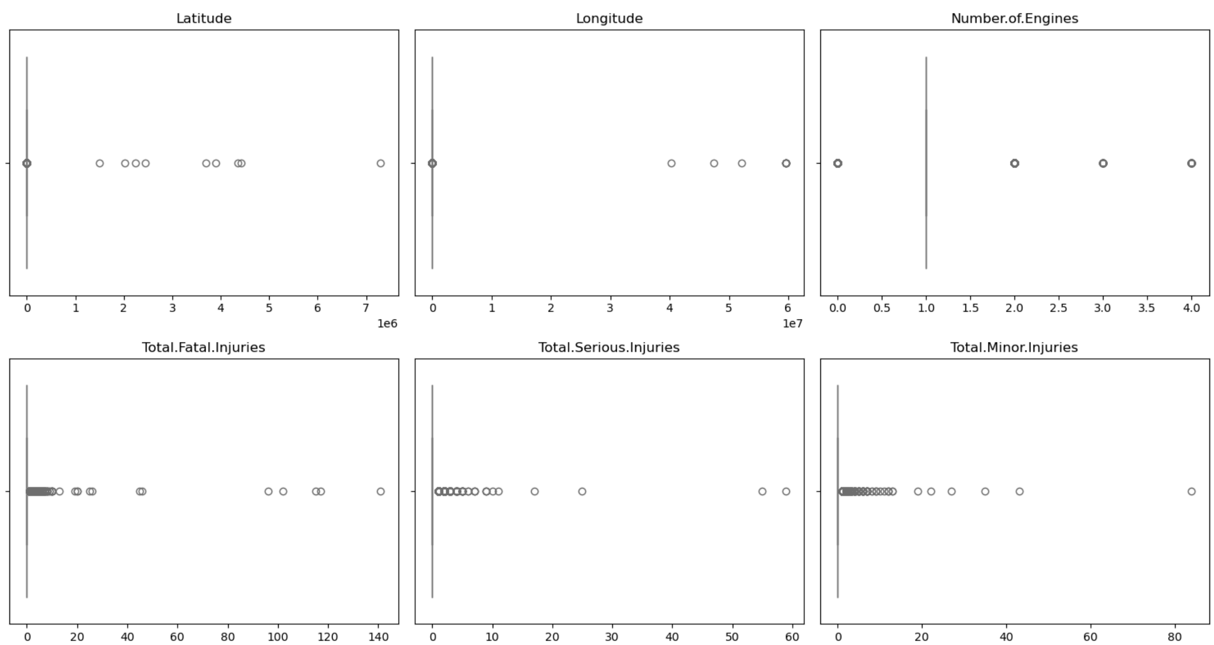
	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Cou
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	U S
5	20170710X52551	Accident	NYC79AA106	1979-09-17	BOSTON, MA	U S
593	20080417X00504	Accident	MIA08CA076	1982-03-16	MOBILE, AL	U S
3654	20051208X01953	Accident	SEA83LA209	1983-01-08	Goldendale, WA	U S
6202	20020904X01525	Accident	SEA83FA208	1983-09-09	Kalispell, MT	U S
22096	20001213X27446	Accident	LAX89LA068	1988-12-23	Midway Islands, PO	U S
24567	20021022X05356	Accident	CHI90LA280	1989-12-01	ENGADINE, MI	U S
26826	20030411X00484	Accident	ANC91GAMS1	1990-10-11	Deadhorse, AK	U S
31353	20170710X10920	Accident	FTW92FA224	1992-09-05	Alpine, TX	U S
38740	20011127X02295	Accident	NYC96FA192	1995-11-28	Marlinton, WV	U S
42691	20001208X08803	Accident	CHI97FA308	1997-09-14	St. Ignaces, MI	U S
44870	20001211X11043	Accident	FTW98FA380	1998-09-11	HOUSTON, TX	U S
45203	20041203X01907	Accident	ATL99FA136	1998-11-04	Robbinsville, NC	U S
45404	20001211X11573	Accident	LAX99FA051	1998-12-17	LOS ANGELES, CA	U S
45592	20001205X00119	Incident	ANC99IA027	1999-02-05	FAIRBANKS, AK	U S

15 rows × 29 columns

In [63]:

```
# check on outliers on the numeric columns
import seaborn as sns
# Select numeric columns
```

```
numeric_cols = df.select_dtypes(include='number').columns

# Set the number of rows and columns for the subplot grid
nrows = 2
ncols = 3

# Create subplots
fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(15, 8))
axes = axes.flatten()  # Flatten  for easy indexing

# Plot each numeric column
for i, col in enumerate(numeric_cols[:nrows * ncols]):  # Limit to fit grid
    sns.boxplot(data=df, x=col, ax=axes[i], color='skyblue')
    axes[i].set_title(f'{col}')
    axes[i].set_xlabel('')

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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



3. **Univariate & Bivariate Analysis**

- Explore the most common aircraft types and count how many incidents each has.
- Analyze incident trends over time to detect rising or declining risk patterns

In [64]:
```
# check the  aircraft type and count how many incidents each has
df['Model'].value_counts().head()
```

Out[64]:
```
Model
152          195
172N         175
172S         154
172          122
PA-28-140    117
Name: count, dtype: int64
```

In [67]:
```
# Analyze incident trends over time to detect rising or declining risk patterns
# Group the model data with the  event year column.
# Change the event date column to real-time dates
df['Event.Date']=pd.to_datetime(df['Event.Date'], errors= 'coerce')

# extract the years from the `df['Event.Date']`
df['Event.Year']= df['Event.Date'].dt.year
```

In [69]:
```
grouped_df=df.groupby(['Event.Year','Model']).size().reset_index(name= 'incident_tt
```

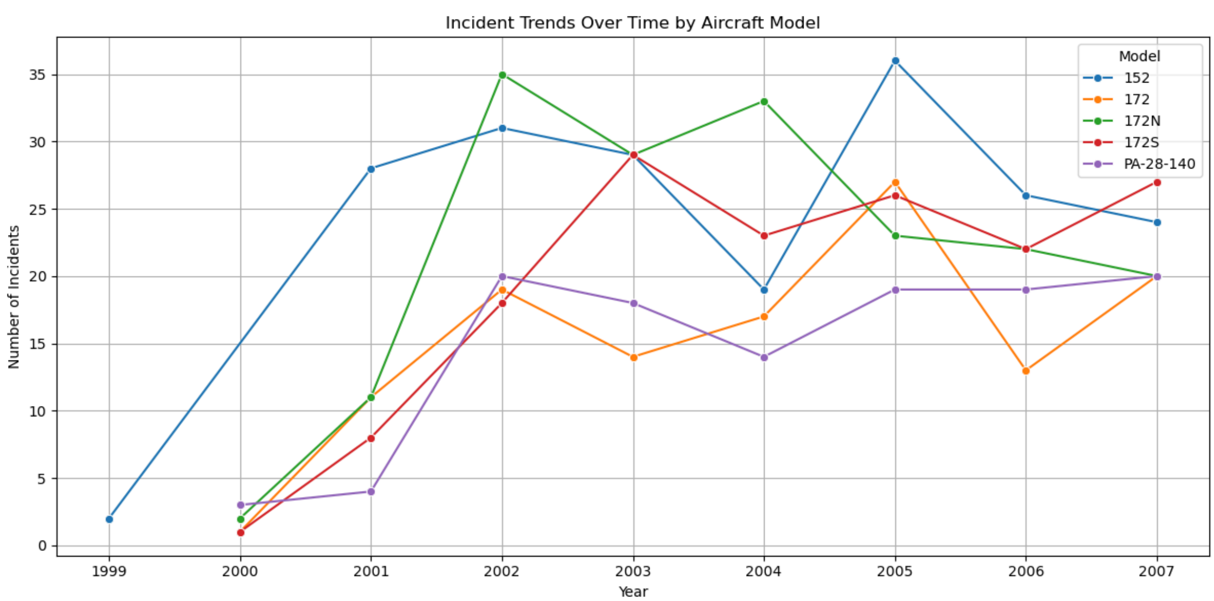
In [71]:
```
top_models = df['Model'].value_counts().head().index
grouped_df = grouped_df[grouped_df['Model'].isin(top_models)]
grouped_df
```

Out[71]:

	Event.Year	Model	incident_ttl
14	1999	152	2
27	2000	172	1
30	2000	172N	2
32	2000	172S	1
127	2000	PA-28-140	3
181	2001	152	28
185	2001	172	11
193	2001	172N	11
197	2001	172S	8
548	2001	PA-28-140	4
703	2002	152	31
713	2002	172	19
725	2002	172N	35
730	2002	172S	18
1287	2002	PA-28-140	20
1548	2003	152	29
1555	2003	172	14
1568	2003	172N	29
1572	2003	172S	29
2214	2003	PA-28-140	18
2519	2004	152	19
2528	2004	172	17
2541	2004	172N	33
2545	2004	172S	23
3133	2004	PA-28-140	14
3396	2005	152	36
3405	2005	172	27
3418	2005	172N	23
3422	2005	172S	26
4039	2005	PA-28-140	19

	Event.Year	Model	incident_ttl
4319	2006	152	26
4326	2006	172	13
4341	2006	172N	22
4345	2006	172S	22
4948	2006	PA-28-140	19
5208	2007	152	24
5215	2007	172	20
5231	2007	172N	20
5235	2007	172S	27
5872	2007	PA-28-140	20

In [73]:
```
# plot the data  for better visualization
plt.figure(figsize=(12, 6))
sns.lineplot(data=grouped_df, x='Event.Year', y='incident_ttl', hue='Model', marker
plt.title('Incident Trends Over Time by Aircraft Model')
plt.xlabel('Year')
plt.ylabel('Number of Incidents')
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [75]:
```
grouped = df.groupby(['Event.Year', 'Model', 'Injury.Severity']).size().reset_index
grouped
```

Out[75]:

	Event.Year	Model	Injury.Severity	Incident_ttl
0	1974	172M	Fatal	1
1	1979	DC9	Minor	1
2	1982	C24R	Fatal	1
3	1983	14-19-3	Fatal	1
4	1983	182N	Fatal	1
...	...	...	...	...
8149	2021	T303	Fatal	1
8150	2021	U206	None	1
8151	2022	182	None	1
8152	2022	M-7-235B	None	1
8153	2022	PA-22-160	None	1

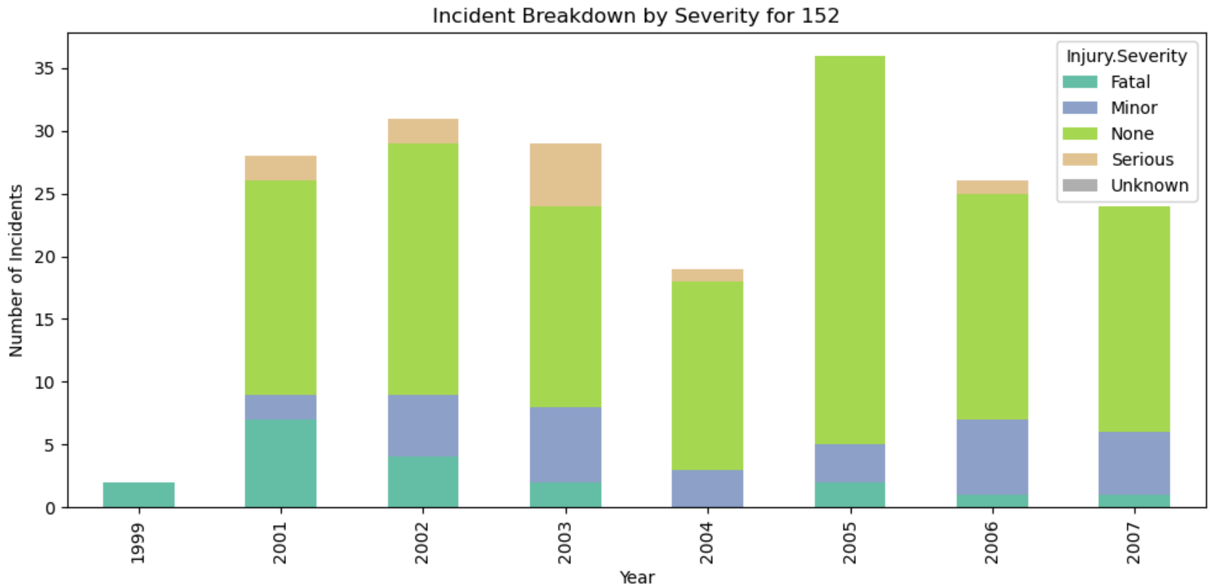
8154 rows × 4 columns

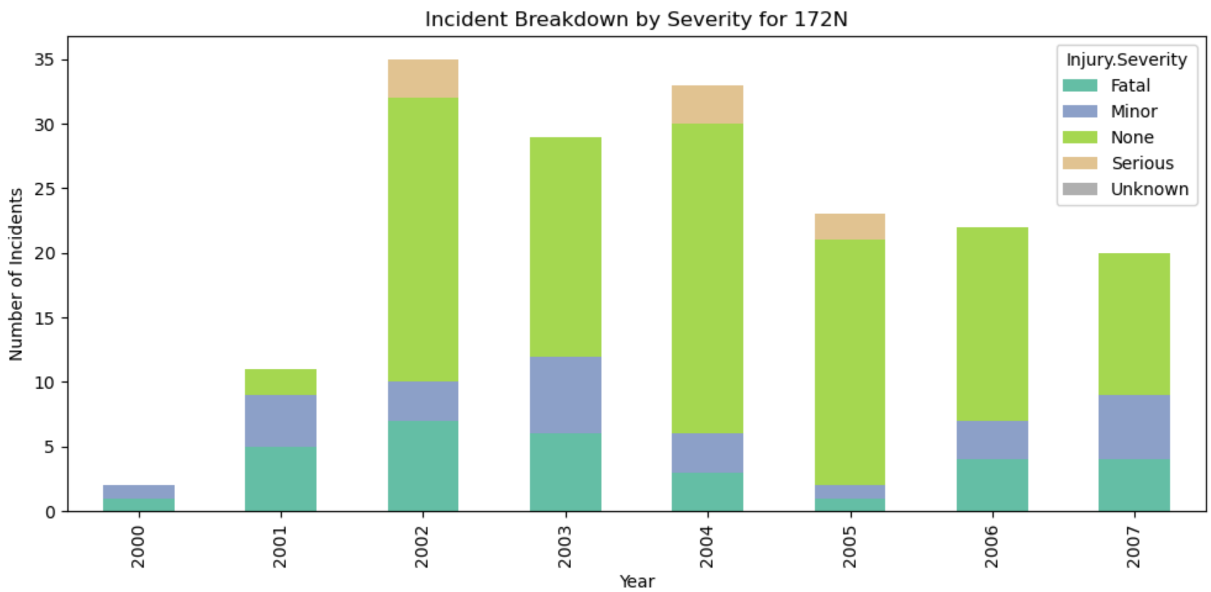
In [77]:

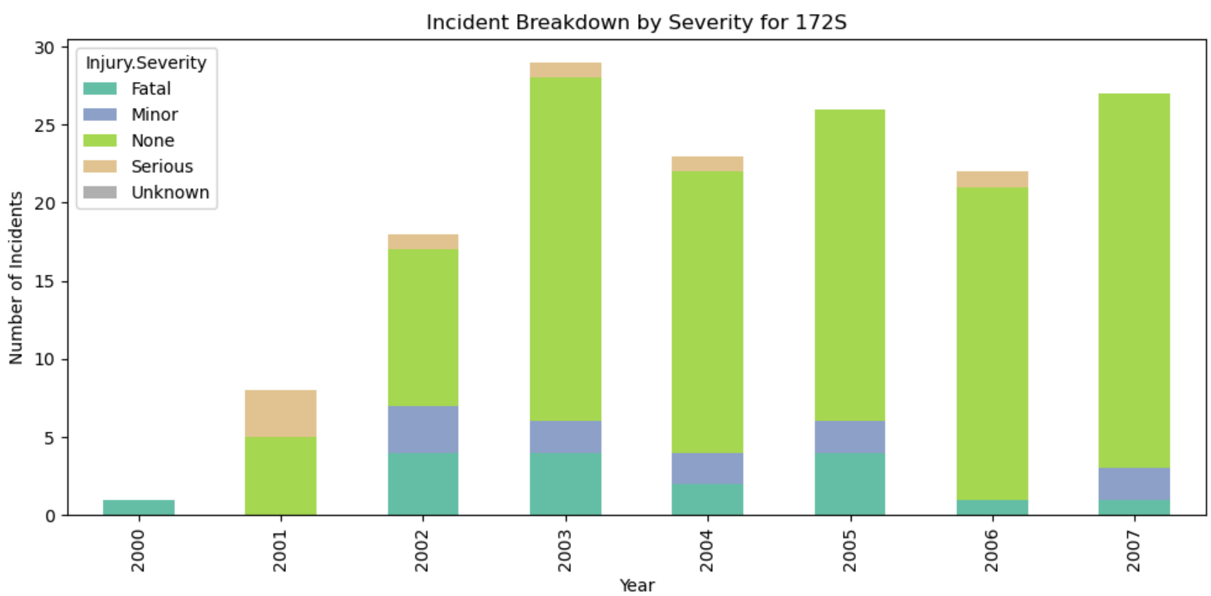
```
# Change this grouped data to a pivot table to be able to plot a grouped bar format
pivot_df = grouped.pivot_table(index=['Event.Year', 'Model'],
                                 columns='Injury.Severity',
                                 values='Incident_ttl',
                                 fill_value=0).reset_index()
```

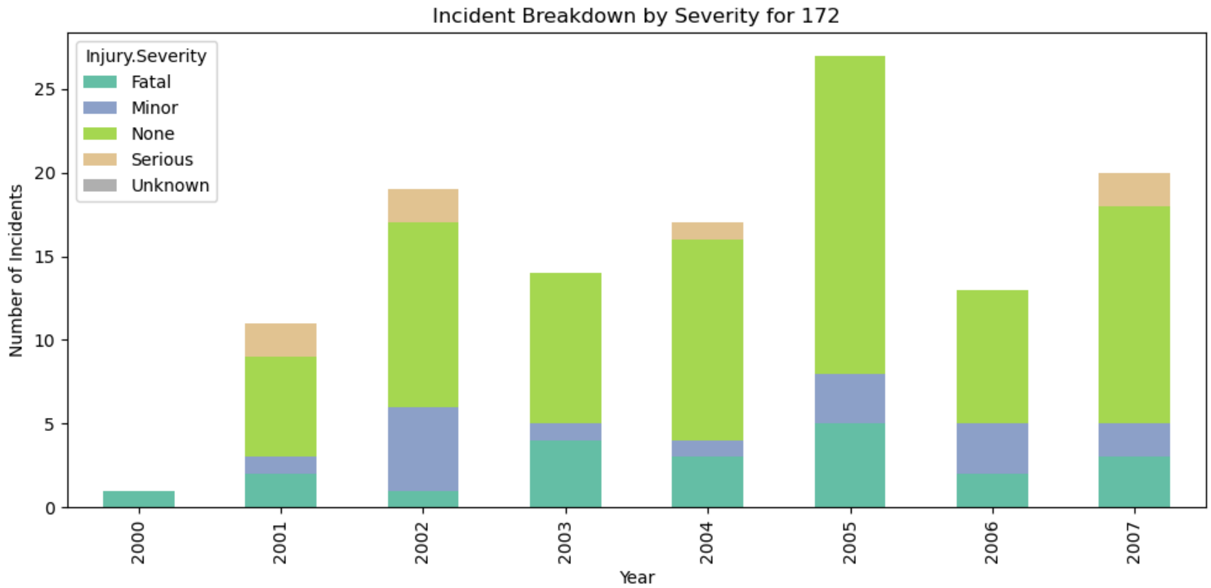
In [79]:

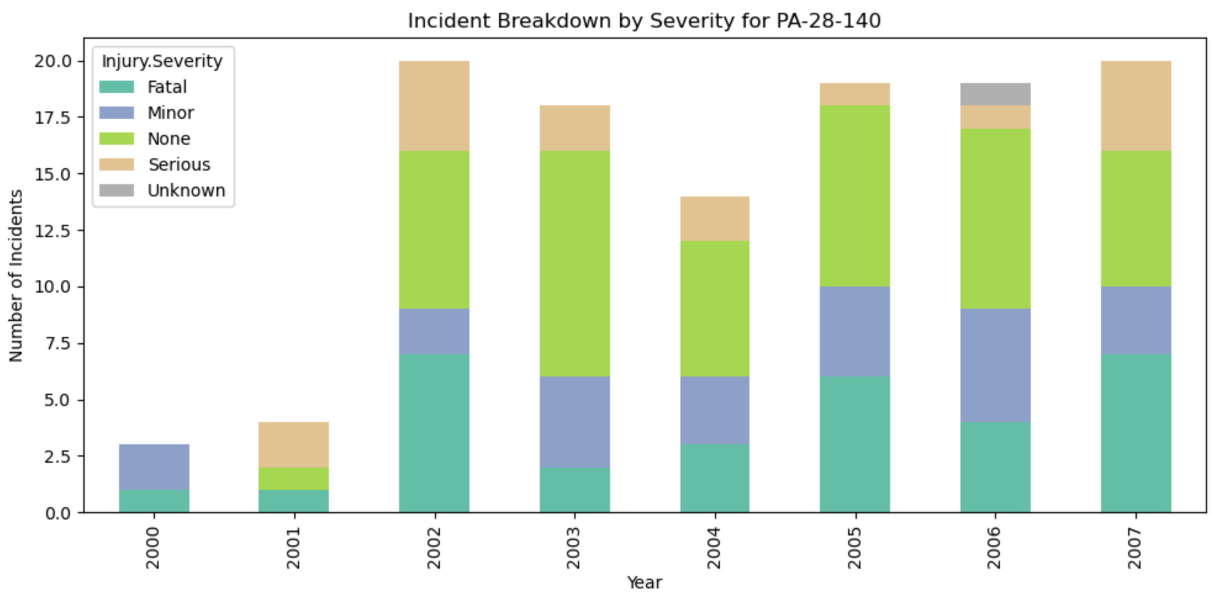
```
# Loop through the top 5 models and plot for each
for model_name in top_models:
    model_data = pivot_df[pivot_df['Model'] == model_name]

    # Plot a stacked bar chart
    model_data.plot(x='Event.Year', kind='bar', stacked=True, figsize=(10, 5), colo
    plt.title(f'Incident Breakdown by Severity for {model_name}')
    plt.ylabel('Number of Incidents')
    plt.xlabel('Year')
    plt.tight_layout()
    plt.show()
```


Incident Breakdown by Severity for 152
Number of Incidents
Year
1999
2001
2002
2003
2004
2005
2006
2007
Injury.Severity
Fatal
Minor
None
Serious
Unknown


Incident Breakdown by Severity for 172N
Number of Incidents
Year
2000
2001
2002
2003
2004
2005
2006
2007
Injury.Severity
Fatal
Minor
None
Serious
Unknown


Incident Breakdown by Severity for 172S
Number of Incidents
Year
2000
2001
2002
2003
2004
2005
2006
2007
Injury.Severity
Fatal
Minor
None
Serious
Unknown





findings.

- The PA-28-140 Aircraft model has the highest constant incidents. It shows a high probability of danger, while the 172S model shows a minimal probability of danger.
- For further model analysis, will conduct on tableau for more insights.

4. **Multivariate & Grouped Insights**

Compare incident frequencies by usage type (commercial vs private), age, or manufacturer.

Changing the aircraft type from `Model` to `Aircraft category` for more simple and easy visualization creations.

In [81]:
```
# retrieve the  columns to be used
df['Aircraft.Category'].value_counts()
```

Out[81]:
```
Aircraft.Category
Unknown Category    8334
Airplane            3084
Helicopter           390
Glider                88
Balloon               40
Gyrocraft             21
Ultralight             6
Blimp                  3
Powered-Lift           2
Name: count, dtype: int64
```

In [83]:
```
# can be used for andvanced analysis of this objective .
df['Purpose.of.flight'].value_counts()
```

Out[83]:
```
Purpose.of.flight
Personal                     7044
Instructional                1559
Unknown                      1188
Aerial Application            483
Business                      403
Positioning                   374
Other Work Use                241
Public Aircraft               149
Flight Test                   124
Aerial Observation             94
Executive/corporate            59
Ferry                          58
Air Race/show                  57
Skydiving                      31
Public Aircraft - Federal      26
Banner Tow                     23
External Load                  17
Public Aircraft - State        14
Glider Tow                     10
Public Aircraft - Local         9
Firefighting                    4
Air Drop                        1
Name: count, dtype: int64
```

In [85]:
```
df['FAR.Description'].value_counts()
```

Out[85]:
```
FAR.Description
Unknown FAR Category              8330
Part 91: General Aviation         3244
Part 137: Agricultural             162
Part 135: Air Taxi & Commuter       85
Part 121: Air Carrier               44
Non-U.S., Non-Commercial            35
Non-U.S., Commercial                22
Public Use                          18
Unknown                              6
Part 129: Foreign                    5
Part 133: Rotorcraft Ext. Load       5
NUSN                                 5
091                                  3
Part 91 Subpart K: Fractional        1
Part 125: 20+ Pax,6000+ lbs          1
Public Aircraft                      1
NUSC                                 1
Name: count, dtype: int64
```

In [87]:
```
df['Make'].value_counts()
```

Out[87]:
```
Make
Cessna                            3485
Piper                             1854
Beech                              751
Bell                               361
Robinson                           308
                                  ...
Davenport (van's)                    1
Dantzer Lawrence L                   1
Aircraft Mfg & Development Co.       1
Murphy                               1
MAULE                                1
Name: count, Length: 1715, dtype: int64
```

In [89]:
```
# Calculate the  Aircraft age.
# limitation: there is no manufacturing Date or Certificate.Issued.Date
# For  temporary trends,  use the `Event.Year` to analyze usage .
```

In [91]:
```
# use the df['FAR.Description'] to create a usage_type column
def classify_usage(far):
    if pd.isna(far):
        return 'Unknown'
    elif '121' in far or '135' in far:
        return 'Commercial'
    elif '91' in far:
        return 'Private'
    else:
        return 'Other'

df['Usage.Type'] = df['FAR.Description'].apply(classify_usage)
```

In [93]:
```
# create  a pivot table to create  incidents by Aircraft Category and usage.
pivot_t1=df.pivot_table(index= 'Aircraft.Category',
```

```
                        columns= 'Usage.Type',
                        values= 'Event.Id',
                        aggfunc='count',
                        fill_value=0
)
pivot_t1
```

Out[93]:

Usage.Type	Commercial	Other	Private
Aircraft.Category			
Airplane	102	218	2764
Balloon	0	0	40
Blimp	0	0	3
Glider	0	1	87
Gyrocraft	0	0	21
Helicopter	27	42	321
Powered-Lift	0	0	2
Ultralight	0	0	6
Unknown Category	0	8330	4

In [95]:

```
# create a pivot table  to gain incidents  by Manufacturer and usage_type
pivot_t2=df.pivot_table(index='Make',
                         columns='Usage.Type',
                         values='Event.Id',
                         aggfunc='count',
                         fill_value=0
).sort_values(by='Commercial', ascending=False)  # arranges  by descending order us
pivot_t2
```

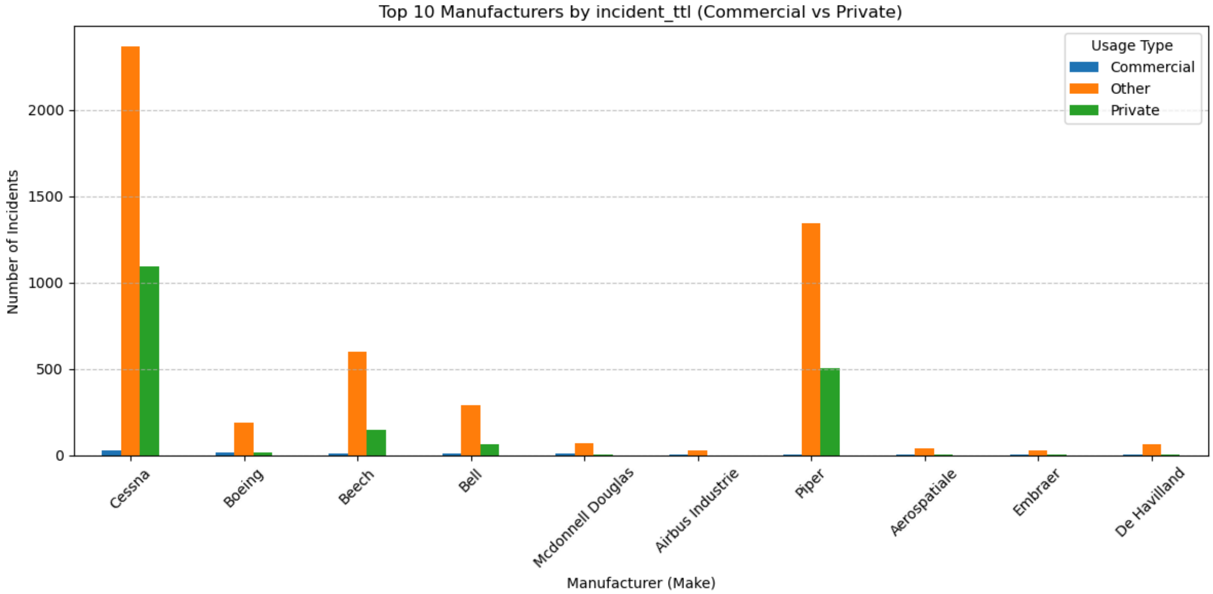
Out[95]:

Usage.Type	Commercial	Other	Private
Make			
Cessna	28	2365	1092
Boeing	16	187	17
Beech	9	597	145
Bell	9	288	64
Mcdonnell Douglas	9	72	3
...	...	...	...
Fox Alfred C.	0	1	0
Found Aircraft Canada Inc	0	0	2
Found Aircraft Canada	0	1	0
Found Acft	0	0	1
Zorn	0	1	0

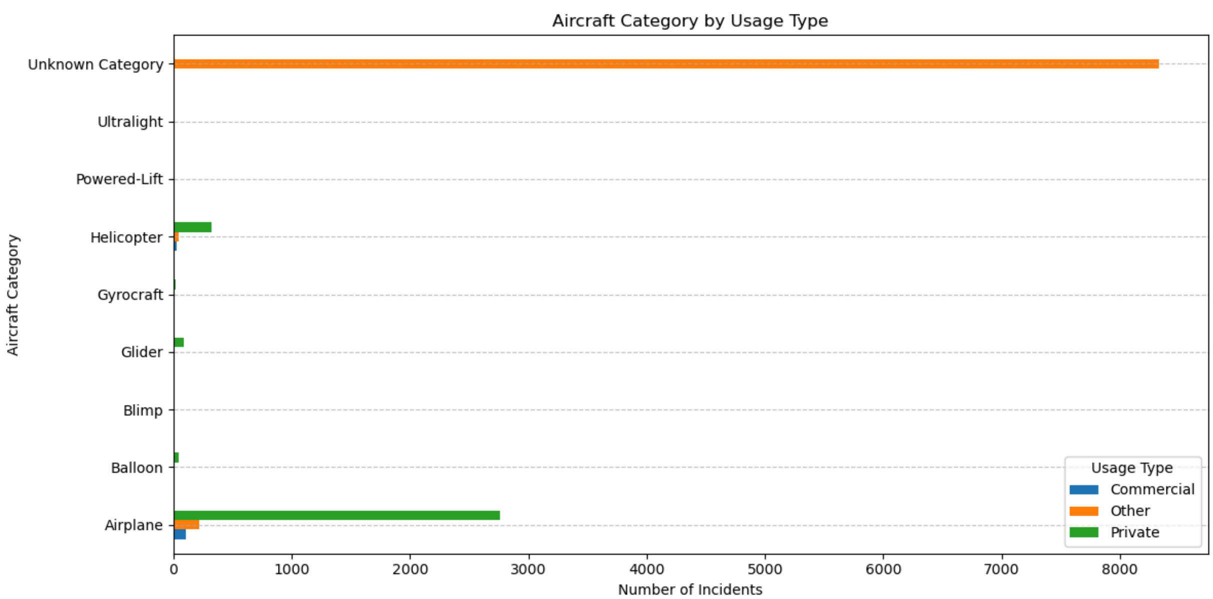
1715 rows × 3 columns

In [97]:

```
# plot the pivot_t2 to  check on the  top ten manufacturers for the  plane models.
Top_10=pivot_t2.head(10)
# plot a grouped bar chart
Top_10.plot(kind='bar', figsize=(12, 6))

# Label the chart
plt.title('Top 10 Manufacturers by incident_ttl (Commercial vs Private)')
plt.ylabel('Number of Incidents')
plt.xlabel('Manufacturer (Make)')
plt.xticks(rotation=45)
plt.legend(title='Usage Type')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

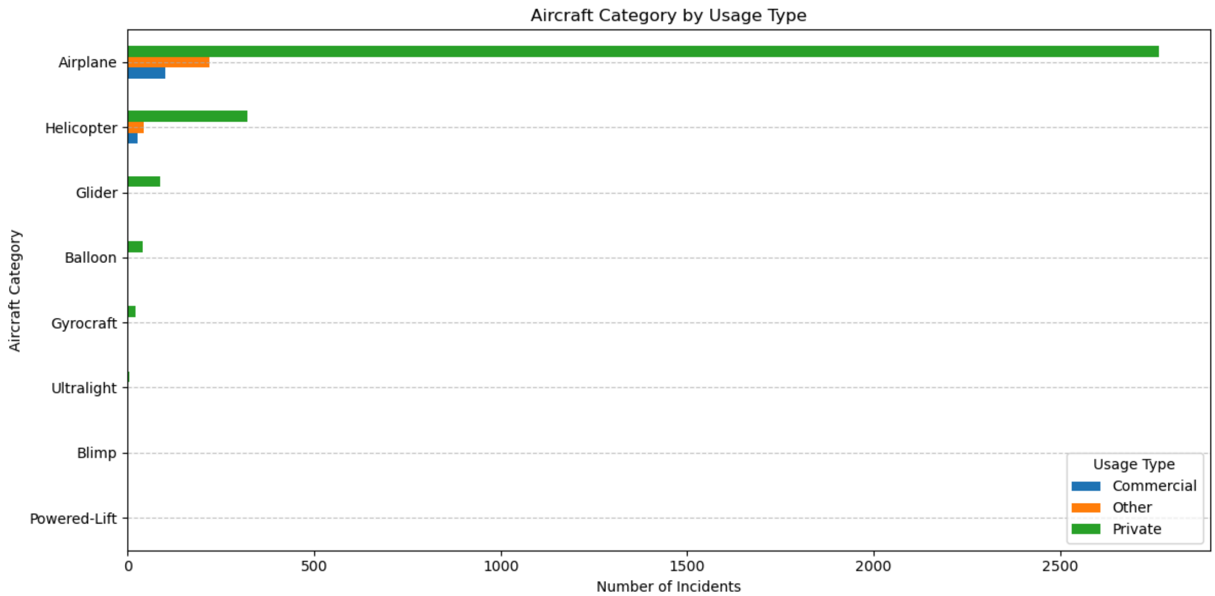


In [99]:
```
#  plot the pivot_t1 to  check on the  top Aircraft category  by usage type.
# plot a  grouped barh graph.
pivot_t1.plot(kind='barh', figsize=(12, 6))
# Label chart
plt.title('Aircraft Category by Usage Type')
plt.xlabel('Number of Incidents')
plt.ylabel('Aircraft Category')
plt.legend(title='Usage Type')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



In [101…
```
# improved graph
# rmove the Unknown cartegory and sort the values to ascending order
pivot_filtered=pivot_t1.drop(index= 'Unknown Category', errors= 'ignore')
pivot = pivot_filtered.sort_values(by=pivot_filtered.columns.to_list(), ascending=T
# plot the data.
pivot.plot(kind='barh', figsize=(12, 6))
```

```
# Label chart
plt.title('Aircraft Category by Usage Type')
plt.xlabel('Number of Incidents')
plt.ylabel('Aircraft Category')
plt.legend(title='Usage Type', loc=4)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



My Findings

- according to the visualizations, private aviation has more incidents than the commercial ones.
- The manufacturers(Make) cessna and piper have high incidents do to their products esspecially on the private aircrafts

5. **Incident Causes & Locations**

- Visualize the top causes of incidents and the regions with the highest incident density.

In [103…
```
df.columns
```

Out[103…
```
Index(['Event.Id', 'Investigation.Type', 'Accident.Number', 'Event.Date',
       'Location', 'Country', 'Latitude', 'Longitude', 'Airport.Name',
       'Injury.Severity', 'Aircraft.damage', 'Aircraft.Category', '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',
       'PubDate_Display', 'Event.Year', 'Usage.Type'],
      dtype='object')
```

In [105…
```
# let's load the recommended columns
df['Broad.phase.of.flight'].value_counts()
```

Out[105…

```
Broad.phase.of.flight
Landing          3215
Takeoff          2195
Cruise           1775
Maneuvering      1475
Approach         1221
Unknown Phase     483
Taxi              377
Descent           343
Climb             339
Go-around         263
Standing          199
Unknown            65
Other              18
Name: count, dtype: int64
```

In [107…

```python
df['Injury.Severity'].value_counts()
```

Out[107…

```
Injury.Severity
None       6337
Fatal      2621
Minor      1716
Serious    1287
Unknown       7
Name: count, dtype: int64
```

In [109…

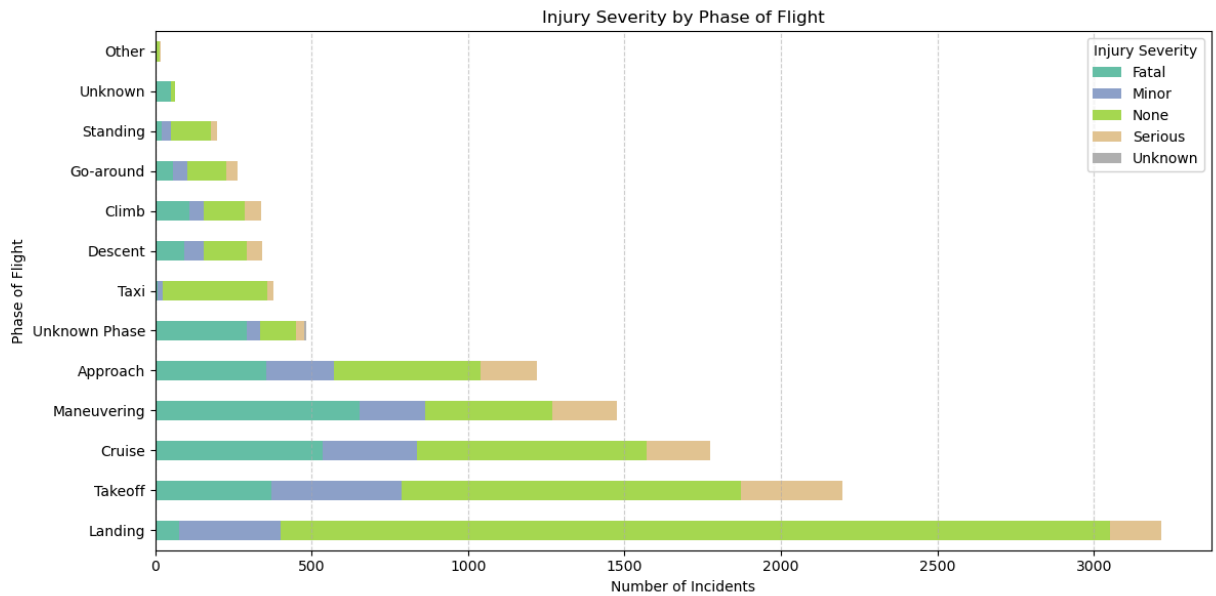
```python
# Group and reshape your clean data
phase_severity = df.groupby(['Broad.phase.of.flight', 'Injury.Severity']).size().un
phase_severity
```

Out[109…

Injury.Severity	Fatal	Minor	None	Serious	Unknown
Broad.phase.of.flight					
Approach	356	214	471	180	0
Climb	108	48	129	54	0
Cruise	534	302	735	204	0
Descent	92	63	139	49	0
Go-around	56	46	126	35	0
Landing	78	322	2650	165	0
Maneuvering	653	210	405	207	0
Other	3	3	8	4	0
Standing	21	28	128	22	0
Takeoff	370	419	1084	322	0
Taxi	7	18	333	19	0
Unknown	51	0	14	0	0
Unknown Phase	292	43	115	26	7

In [111…

```
# Let's plot the two columns to plot a grouped bar
# Sort by total incident count descending
phase_severity = phase_severity.loc[phase_severity.sum(axis=1).sort_values(ascendin
# Plot
phase_severity.plot(kind='barh', stacked=True, figsize=(12, 6), colormap='Set2')

# Labels and formatting
plt.title('Injury Severity by Phase of Flight')
plt.xlabel('Number of Incidents')
plt.ylabel('Phase of Flight')
plt.legend(title='Injury Severity')
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



My Finding

- The phase of flight with high incidents is landing.

5. **Insights and Recommendations**

- Other and Private usage types of aircraft have higher injury severity than the commercial ones. This suggests that general aviation poses a higher volume of safety incidents, possibly due to looser regulations, pilot experience, or maintenance variation.

- The Aircraft Category poses high incidents, most especially in private/personal aviation. However, the multi-engine aircraft categories, such as Gyrocraft, Powered-Lift, and Ultralight, have had slight incidents, but these aircraft can be suitable for starting a business venture in aviation.

- While the landing phase accounts for the highest number of incidents, most result in non-injury outcomes. In contrast, the maneuvering, takeoff, and cruise phases—though less frequent—are associated with high injury severity. This suggests a need to shift to a safety focus on in-flight operations, where risk per incident may be higher.

6. **Challenges and conclusions**

- The data had a lot of critical missing values.

In [115…

```
# lets change the data back to csv for Tableau analysis.
df.to_csv('Aviation_clean_data.csv', index=False)
```