Phase-1-Project-Aircraft-Accident-Analysis / Phase_1_Project.ipynb

2225 lines (2225 loc) · 256 KB

Business Understanding/Overview

I am charged with determining which aircrafts have the lowest risk for the company to start a new business since it is trying to expand and diversify its portfolio. They are interested in purchasing and operating airplanes for both commercial and private enterprises, but do not know anything about the potential risks of aircraft. My aim is to assist look at the data, analyse then translate my findings into actionable insights that the head of the new aviation division can use to help decide which aircraft to purchase

Problem Statement

The company wants to get into new industries and explore aircrafts for commercial and private purposes and it requires assessment form the data to identify low-risk aircraft. The goal is to recommend low risk aircraft that are suitable for successful market entry. Actionable insights will guide the aviation division in making informed purchasing decisions.

Objective

1. Analyze past data to identify accident trends over time and determine whether accident rates are improving or worsening

2. Identify and compare accident rates versus aircraft model to find the ones with the lowest accident rate and the safest

3. Look at the location with most accidents to identify regions or routes with higher risk which will help in planning and strategic deployment of the aircraft.

Success Criteria

1. The project will be successful if we are able to identify how viable it is to get into the aircraft business through the analysis made

2. If we are able to deliver clear and analysed accident rates created by specific aircrafts to identify the one with the lowest risk

3. When we are able to identify the suitable location for aircraft deployment

Limitations and Assumptions

- Due to missing and incomplete data, we may not get the true picture of the recommendations that would be made
- We only have Accidents data and that may not be conclusive in determining whether an aircraft is viable, durable and cost efficient. Safety of the aircraft alone might not be sufficient

Data Understanding

In [4]:

```
#Imporrt Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:

```
#Reading the dataset from a csv file using pandas
#I encountered an error and researched on how to fix it with the below code

df = pd.read_csv('AviationData.csv', encoding='latin1', low_memory=False)
```

Aviation Data

The NTSB aviation accident database contains information from 1962 and later about civil aviation accidents and selected incidents within the United States, its territories and possessions, and in international waters.

In [76]:

```
#Looking at head to see how the  columns and rows look like
df.head()
```

Out[76]:

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

5 rows × 27 columns

5 rows × 27 columns

In [7]:

```
df.tail()
```

Out[7]:

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

5 rows × 31 columns

In [8]:

```
#Checking for shape of the dataset
df.shape
```

Out[8]: (88889, 31)

In [10]:

```
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
```

```
 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
```

1. There are 31 columns, including categorical and numerical data, with several columns containing missing values.
2. Most columns are of type object.
3. Columns such as Event.Date, Publication.Date, and Latitude/Longitude may require: Conversion to appropriate types ie, datetime for date columns, float for geospatial coordinates.

In [11]:

```
#Checking for any duplicates
df.duplicated().sum()
```

Out[11]: 0

In [ ]:

```
#Looking at the numerical features
df.describe()
```

Out[ ]:

	Number.of.Engines	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries
count	82805.000000	77488.000000	76379.000000	76956.000000
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

In [13]:

```
#Checking for null values
df.isna().sum()
```

Out[13]:

```
Event.Id                      0
Investigation.Type            0
Accident.Number               0
Event.Date                    0
Location                     52
Country                     226
Latitude                  54507
Longitude                 54516
Airport.Code              38640
Airport.Name              36099
Injury.Severity            1000
Aircraft.damage            3194
Aircraft.Category         56602
Registration.Number        1317
Make                         63
Model                        92
Amateur.Built               102
Number.of.Engines          6084
Engine.Type                7077
FAR.Description           56866
Schedule                  76307
Purpose.of.flight          6192
Air.carrier               72241
Total.Fatal.Injuries      11401
Total.Serious.Injuries    12510
Total.Minor.Injuries      11933
Total.Uninjured            5912
Weather.Condition          4492
Broad.phase.of.flight     27165
Report.Status              6381
Publication.Date          13771
dtype: int64
```

I discovered that there were numerous missing values in the dataset that needed cleaning. To manage this efficiently, I first calculated the percentage of missing values to assist the analysis process.

In [14]:

```python
# Calculating the missing values
missing_summary = df.isna().sum().sort_values(ascending=False)
missing_percentage = (df.isna().mean() * 100).sort_values(ascending=False)

# Creating and displaying summary DataFrame
missing_data = pd.DataFrame({
    'Missing Count': missing_summary,
    'Missing %': missing_percentage.round(2)
})

# Displaying the columns with the missing values
missing_data = missing_data[missing_data['Missing Count'] > 0]

missing_data
```

Out[14]:

	Missing Count	Missing %
Schedule	76307	85.85
Air.carrier	72241	81.27
FAR.Description	56866	63.97

FAR.Description	56866	63.97
Aircraft.Category	56602	63.68
Longitude	54516	61.33
Latitude	54507	61.32
Airport.Code	38640	43.47
Airport.Name	36099	40.61
Broad.phase.of.flight	27165	30.56
Publication.Date	13771	15.49
Total.Serious.Injuries	12510	14.07
Total.Minor.Injuries	11933	13.42
Total.Fatal.Injuries	11401	12.83
Engine.Type	7077	7.96
Report.Status	6381	7.18
Purpose.of.flight	6192	6.97
Number.of.Engines	6084	6.84
Total.Uninjured	5912	6.65
Weather.Condition	4492	5.05
Aircraft.damage	3194	3.59
Registration.Number	1317	1.48
Injury.Severity	1000	1.12
Country	226	0.25
Amateur.Built	102	0.11
Model	92	0.10
Make	63	0.07
Location	52	0.06

Handling the missing value

Data Cleaning

The following columns were dropped due to the high percentage of missing values:

- **Schedule (85.85% missing)**:
- **Air.carrier (81.27% missing)**:
- **FAR.Description (63.97% missing)**:

- **FAR.Description (63.97% missing)**:
- **Aircraft.Category (63.68% missing)**:
- **Longitude and Latitude (61.33% and 61.28% missing, respectively)**:

In [15]:
```
columns_to_drop = ["Schedule", "Air.carrier", "FAR.Description", "Aircraft.Cat

# Dropping the columns
df = df.drop(columns=columns_to_drop)

# Verifying the columns have been dropped
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88889 entries, 0 to 88888
Data columns (total 25 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   Airport.Code            50249 non-null  object 
 7   Airport.Name            52790 non-null  object 
 8   Injury.Severity         87889 non-null  object 
 9   Aircraft.damage         85695 non-null  object 
 10  Registration.Number     87572 non-null  object 
 11  Make                    88826 non-null  object 
 12  Model                   88797 non-null  object 
 13  Amateur.Built           88787 non-null  object 
 14  Number.of.Engines       82805 non-null  float64
 15  Engine.Type             81812 non-null  object 
 16  Purpose.of.flight       82697 non-null  object 
 17  Total.Fatal.Injuries    77488 non-null  float64
 18  Total.Serious.Injuries  76379 non-null  float64
 19  Total.Minor.Injuries    76956 non-null  float64
 20  Total.Uninjured         82977 non-null  float64
 21  Weather.Condition       84397 non-null  object 
 22  Broad.phase.of.flight   61724 non-null  object 
 23  Report.Status           82508 non-null  object 
 24  Publication.Date        75118 non-null  object 
dtypes: float64(5), object(20)
memory usage: 17.0+ MB
```

Columns with missing values that can be inputed

Location (0.06%)

In [16]:
```
# Checking rows where Location is missing
missing_location_rows = df[df['Location'].isna()]

# Filling missing Location values with Airport.Name
```

```
df['Location'] = df['Location'].fillna(df['Airport.Name'])
```

In [17]:

```
remaining_missing_location = df['Location'].isna().sum()
remaining_missing_location
remaining_missing_location = df['Location'].isna().sum()
remaining_missing_location
```

Out[17]: 50

I decided to drop the remaining 50 missing location entries because imputing them using the mode could introduce inaccuracies. Also, 50 represents only a small percentage of the dataset.

In [18]:

```
# Dropping rows where Location is missing
df = df.dropna(subset=['Location'])
```

In [19]:

```
# Verifying if all missing values in 'Location' are handled
remaining_missing_location = df['Location'].isna().sum()
remaining_missing_location
```

Out[19]: 0

Country (0.25% missing)

In [20]:

```
# Drop rows with missing values in the 'Country' column
dropped_missing_country = df.dropna(subset=['Country'])

# Verify if missing values were removed
remaining_missing_country = dropped_missing_country['Country'].isna().sum()
remaining_missing_country
```

Out[20]: 0

The percentage of the missing rows in the data is insignificant and replacing it might affect the accuracy of the data

Injury.Severity (1.13% missing)

In [77]:

```
# Group by related columns like 'Aircraft.damage' and 'Broad.phase.of.flight'
# Imputing missing 'Injury.Severity' based on the most frequent value in each

grouped_injury_severity = df.groupby(['Aircraft.damage', 'Broad.phase.of.fligh
grouped_injury_severity

# Apply imputation to missing 'Injury.Severity'
for group, mode_value in grouped_injury_severity.items():
    mask = (df['Aircraft.damage'] == group[0]) & (df['Broad.phase.of.flight']
    df.loc[mask & df['Injury.Severity'].isna(), 'Injury.Severity'] = mode_valu

# If there are still missing values, impute using the mode of the 'Injury.Seve
```

```
df['Injury.Severity'] = df['Injury.Severity'].fillna(df['Injury.Severity'].mod
```

In [22]:

```
remaining_missing_injury_severity = df['Injury.Severity'].isna().sum()

remaining_missing_injury_severity
```

Out[22]: 0

Aircraft.damage (3.59% missing)

In [23]:

```
# Impute missing values in 'Aircraft.damage' based on 'Injury.Severity'
# Group by 'Injury.Severity' and calculate the mode of 'Aircraft.damage' for e
grouped_aircraft_damage = df.groupby('Injury.Severity')['Aircraft.damage'].app

# Apply imputation to missing 'Aircraft.damage'
for injury_severity, mode_value in grouped_aircraft_damage.items():
    mask = df['Injury.Severity'] == injury_severity
    df.loc[mask & df['Aircraft.damage'].isna(), 'Aircraft.damage'] = mode_valu

# If there are still missing values, impute using the mode of 'Aircraft.damage
df['Aircraft.damage'] = df['Aircraft.damage'].fillna(df['Aircraft.damage'].mod

# Verify if all missing values in 'Aircraft.damage' are handled
remaining_missing_aircraft_damage = df['Aircraft.damage'].isna().sum()

remaining_missing_aircraft_damage
```

Out[23]: 0

Dropping the two columns since I did not foresee using them in the analysis

In [24]:

```
columns_to_drop = ['Number.of.Engines', 'Engine.Type']

# Dropping the columns
df = df.drop(columns=columns_to_drop)

# Verifying the columns have been dropped
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 88839 entries, 0 to 88888
Data columns (total 23 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Event.Id                88839 non-null  object
 1   Investigation.Type      88839 non-null  object
 2   Accident.Number         88839 non-null  object
 3   Event.Date              88839 non-null  object
 4   Location                88839 non-null  object
 5   Country                 88614 non-null  object
 6   Airport.Code            50249 non-null  object
 7   Airport.Name            52790 non-null  object
 8   Injury.Severity         88839 non-null  object
```

```
 9   Aircraft.damage         88839 non-null  object 
 10  Registration.Number     87544 non-null  object 
 11  Make                    88776 non-null  object 
 12  Model                   88747 non-null  object 
 13  Amateur.Built           88741 non-null  object 
 14  Purpose.of.flight       82655 non-null  object 
 15  Total.Fatal.Injuries    77452 non-null  float64
 16  Total.Serious.Injuries  76345 non-null  float64
 17  Total.Minor.Injuries    76923 non-null  float64
 18  Total.Uninjured         82936 non-null  float64
 19  Weather.Condition       84352 non-null  object 
 20  Broad.phase.of.flight   61712 non-null  object 
 21  Report.Status           82458 non-null  object 
 22  Publication.Date        75080 non-null  object 
dtypes: float64(4), object(19)
memory usage: 16.3+ MB
```

Purpose.of.flight (7.11% missing)

In [25]:

```python
# Drop rows with missing values in the 'Purpose.of.flight' column
Purpose_of_flight_cleaned = df.dropna(subset=['Purpose.of.flight'])

# Verify if any missing values remain in 'Purpose.of.flight'
remaining_missing_purpose_of_flight = Purpose_of_flight_cleaned['Purpose.of.fl

remaining_missing_purpose_of_flight
```

Out[25]: 0

Weather.Condition (5.06% missing)

In [34]:

```python
# Drop rows with missing values in the 'Weather.Condition' column
aviation_data_cleaned = df.dropna(subset=['Weather.Condition'])

# Verify if any missing values remain in 'Weather.Condition'
remaining_missing_weather_condition = aviation_data_cleaned['Weather.Condition

remaining_missing_weather_condition
```

Out[34]: 0

Data Preparation

Columns such as Event.Date, Publication.Date, need to be converted to appropriate types ie, datetime for date columns

In [ ]:

```python
# Convert 'Event.Date' and 'Publication.Date' columns to datetime
df['Event.Date'] = pd.to_datetime(df['Event.Date'], errors='coerce')
df['Publication.Date'] = pd.to_datetime(df['Publication.Date'], errors='coerce

# Check if the changes have been effected
```

```
df.dtypes[['Event.Date', 'Publication.Date']]
```

Out[ ]:
```
Event.Date          datetime64[ns]
Publication.Date    datetime64[ns]
dtype: object
```

In [35]:
```
#Take a look at the columns
print(df.columns)
```

```
Index(['Event.Id', 'Investigation.Type', 'Accident.Number', 'Event.Date',
       'Location', 'Country', 'Airport.Code', 'Airport.Name',
       'Injury.Severity', 'Aircraft.damage', 'Registration.Number', 'Make',
       'Model', 'Amateur.Built', 'Purpose.of.flight', 'Total.Fatal.Injuries',
       'Total.Serious.Injuries', 'Total.Minor.Injuries', 'Total.Uninjured',
       'Weather.Condition', 'Broad.phase.of.flight', 'Report.Status',
       'Publication.Date'],
      dtype='object')
```

Exploratory Data Analysis (EDA)

Performing analysis on the cleaned data

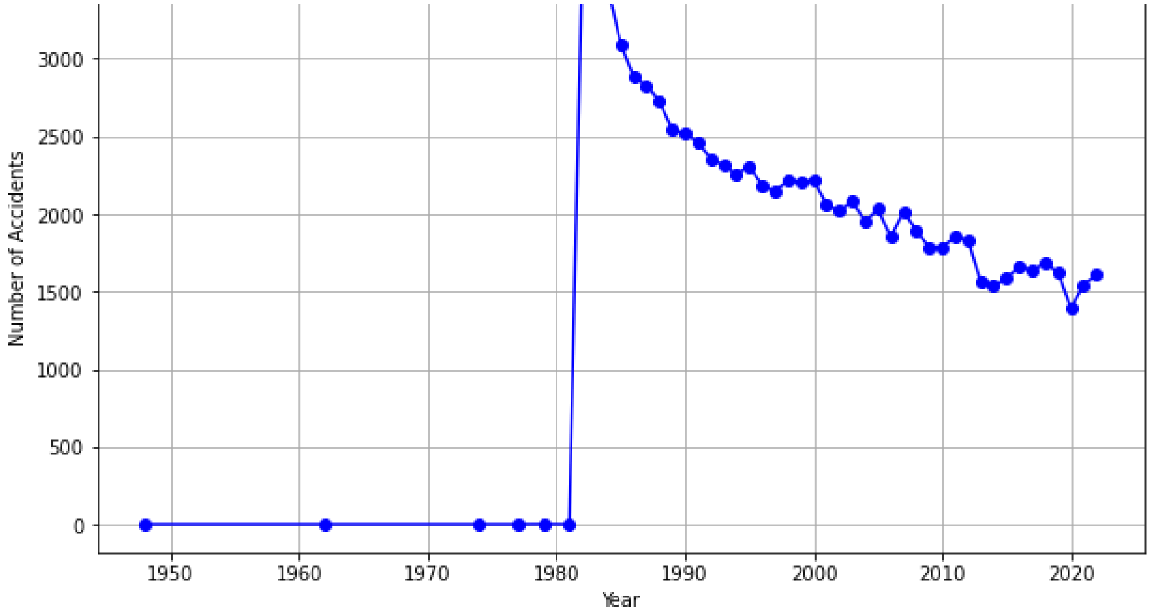
Analyze past data to identify accident trends over time and determine whether accident rates are improving or worsening

Extract Time-Based Features: You can extract different time-based features such as:

Year Month Day Day of the week Quarter This allows for a deeper understanding of how accidents are distributed over time.

In [42]:
```
# Extract year, month, and quarter from 'Event.Date'
df['Year'] = df['Event.Date'].dt.year
df['Month'] = df['Event.Date'].dt.month
df['Quarter'] = df['Event.Date'].dt.quarter
df['DayOfWeek'] = df['Event.Date'].dt.dayofweek
```

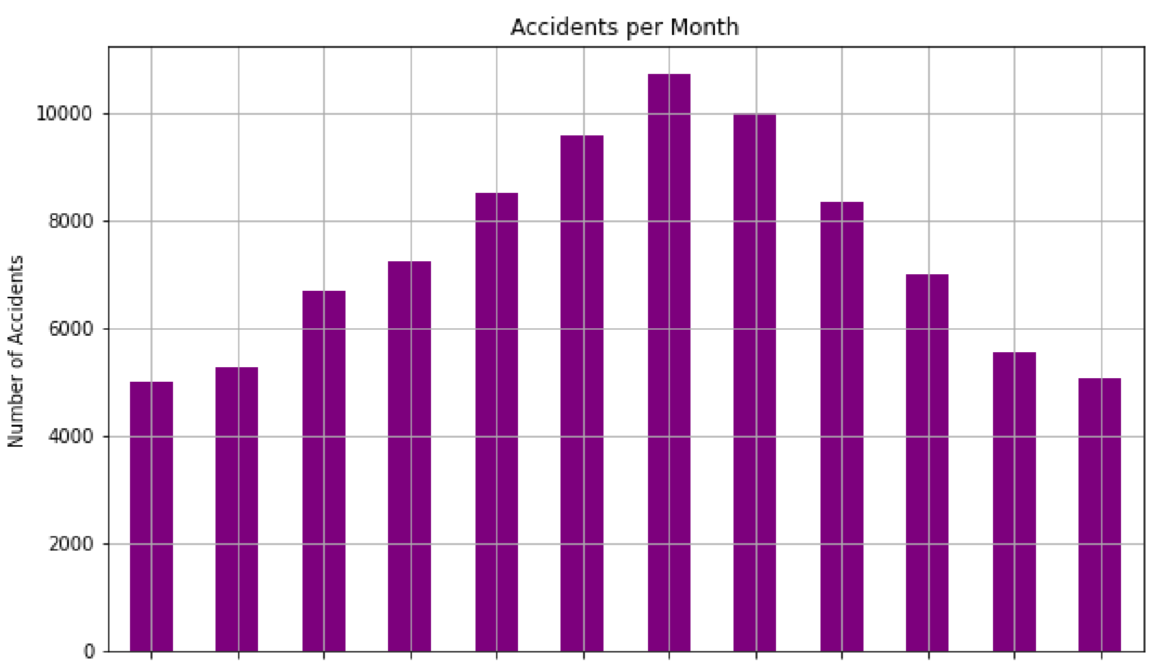
In [43]:
```
# Group by 'Year' to count the number of accidents per year
accidents_per_year = df.groupby('Year').size()

# Plot the trend of accidents over time
plt.figure(figsize=(10, 6))
accidents_per_year.plot(kind='line', marker='o', color='b')
plt.title('Accidents per Year')
plt.xlabel('Year')
plt.ylabel('Number of Accidents')
plt.grid(True)
plt.show()
```



Up until 1980, the number of accidents remained stable, showing little to no change. This could suggest that either accidents were not being recorded or that there were fewer airlines in operation during that time. An unusual event happened in 1980 which caused a spike but the accident rates started improving significantly

In [78]:

```
# Group by month to analyze the seasonality of accidents
accidents_per_month = df.groupby('Month').size()

# Plot the accident distribution across months
plt.figure(figsize=(10, 6))
accidents_per_month.plot(kind='bar', color='purple')
plt.title('Accidents per Month')
plt.xlabel('Month')
plt.ylabel('Number of Accidents')
plt.grid(True)
plt.show()
```

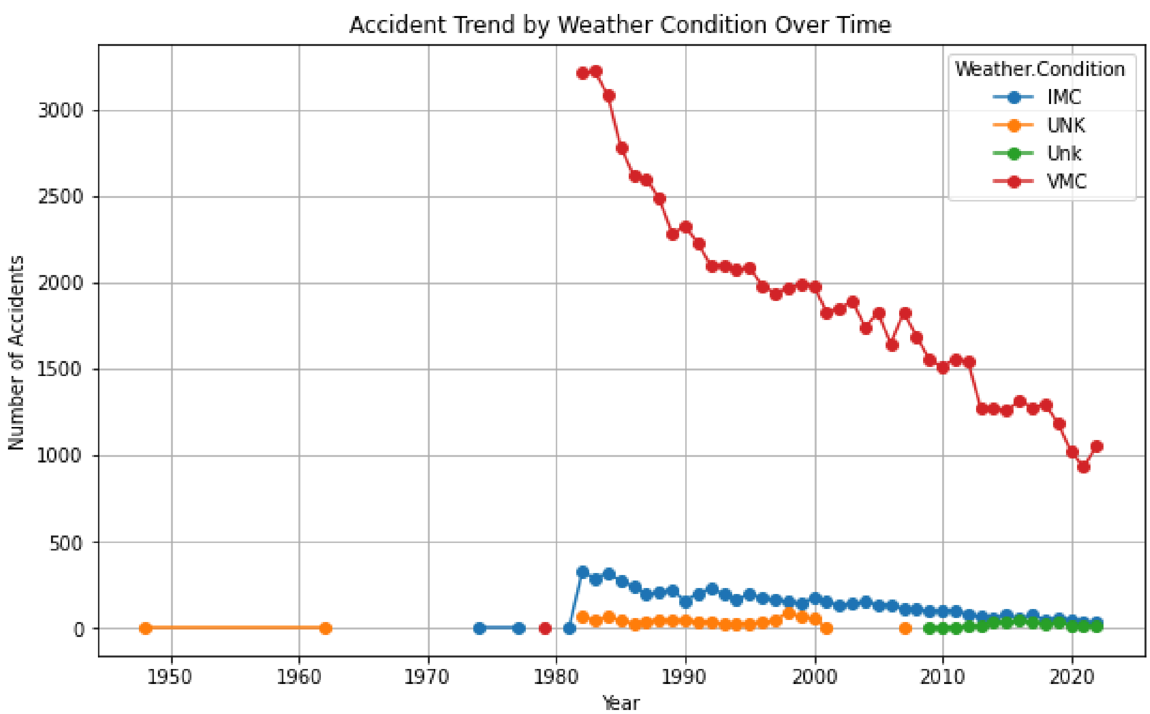


Trying to see how many accidents have been happening per month

The highest accidents took place in july. We need to investigate why

In [47]:

```
# Correlation between accident count and other categorical variables (e.g., we
accident_weather = df.groupby(['Year', 'Weather.Condition']).size().unstack()
accident_weather.plot(kind='line', marker='o', figsize=(10, 6))
plt.title('Accident Trend by Weather Condition Over Time')
plt.xlabel('Year')
plt.ylabel('Number of Accidents')
plt.grid(True)
plt.show()
```



The line plot shows the trend of accidents categorized by weather conditions over time. By grouping the data by year and weather condition, we see how accidents correlate with different weather types in each year. The graph allows us to analyze whether specific weather conditions have influenced accident rates in particular years, helping to identify any patterns or trends related to weather conditions

Identify and compare accident rates versus aircraft model to find the ones with the lowest accident rate and the safest

In [36]:

```
#Retrieving the different aircraft models available
df['Model']
```

Out[36]:

```
0          108-3
1       PA24-180
2           172M
```

```
3              112
4              501
          ...
88884    PA-28-151
88885         7ECA
88886        8GCBC
88887         210N
88888    PA-24-260
Name: Model, Length: 88839, dtype: object
```

In [29]:

```python
#Retrieving the unique values from the 'Model'
df['Model'].unique()
```

Out[29]:

```
array(['108-3', 'PA24-180', '172M', ..., 'ROTORWAY EXEC 162-F',
       'KITFOX S5', 'M-8 EAGLE'], dtype=object)
```

In [37]:

```python
#Getting to know the count of unique values there are each model
df['Model'].value_counts()
```

Out[37]:

```
152                     2365
172                     1755
172N                    1164
PA-28-140                932
150                      829
                        ...
BE-80                      1
M4-220                     1
AVID AIRCRAFT FLYER        1
Classic Sport S-18         1
ACAPELLA                   1
Name: Model, Length: 12309, dtype: int64
```

In [38]:

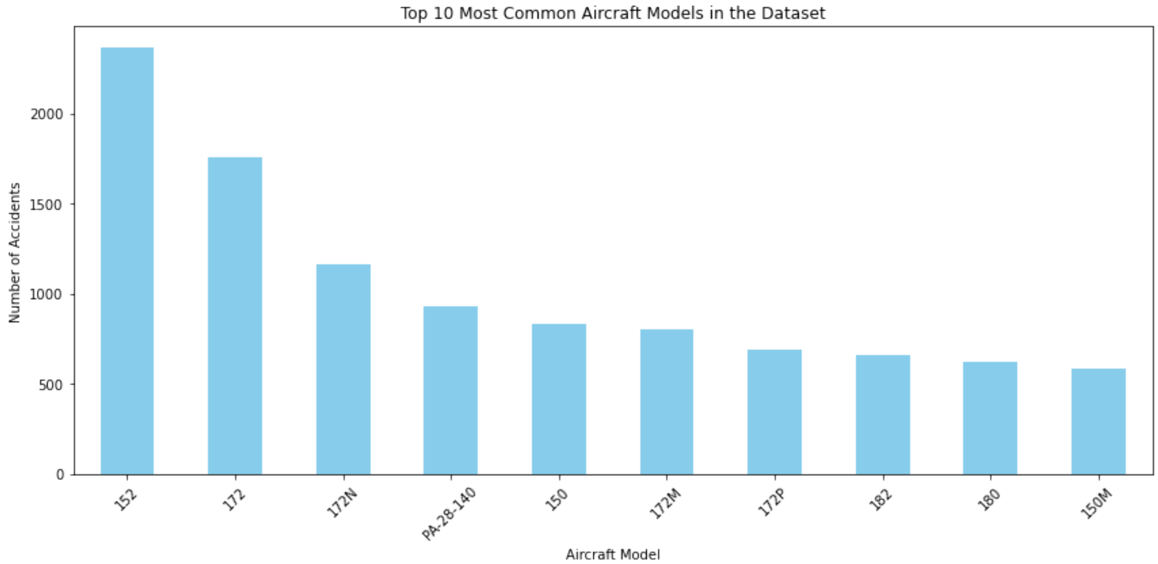
```python
# Get the top 10  most common models
top_models = df['Model'].value_counts().head(10)
top_models
```

Out[38]:

```
152          2365
172          1755
172N         1164
PA-28-140     932
150           829
172M          798
172P          689
182           659
180           622
150M          585
Name: Model, dtype: int64
```

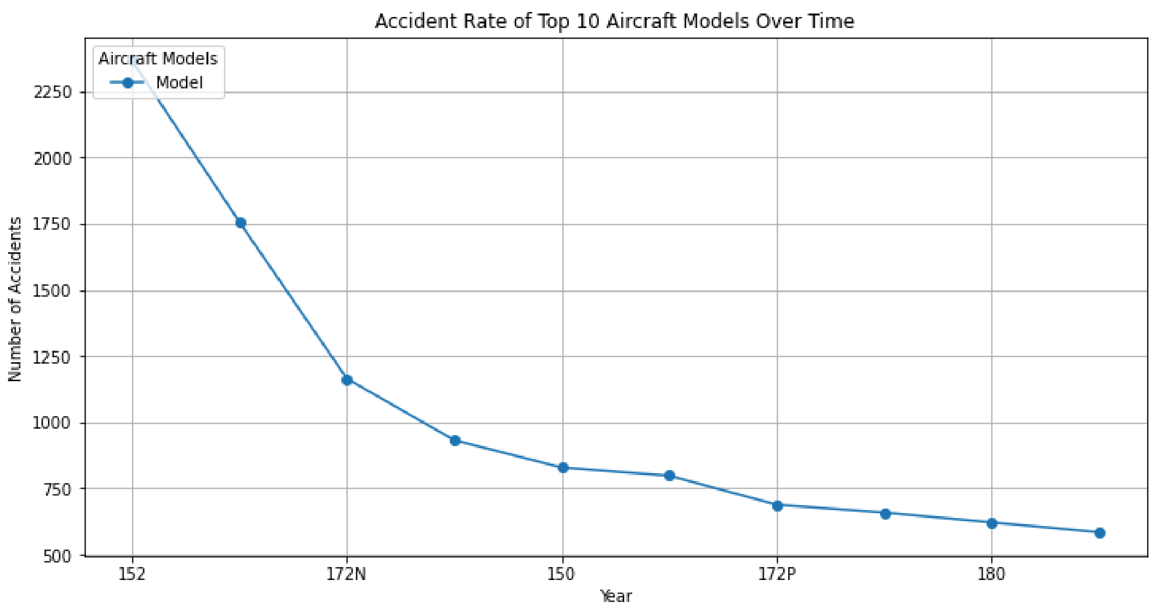
In [ ]:

```python
# Create a bar chart
plt.figure(figsize=(12, 6))
top_models.plot(kind='bar', color='skyblue')
plt.title('Top 10 Most Common Aircraft Models in the Dataset')
plt.xlabel('Aircraft Model')
plt.ylabel('Number of Accidents')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [61]:

```
plt.figure(figsize=(12, 8))
top_models.plot(kind='line', marker='o', figsize=(12, 6))
plt.title('Accident Rate of Top 10 Aircraft Models Over Time')
plt.xlabel('Year')
plt.ylabel('Number of Accidents')
plt.legend(title='Aircraft Models', loc='upper left')
plt.grid(True)
plt.show()
```



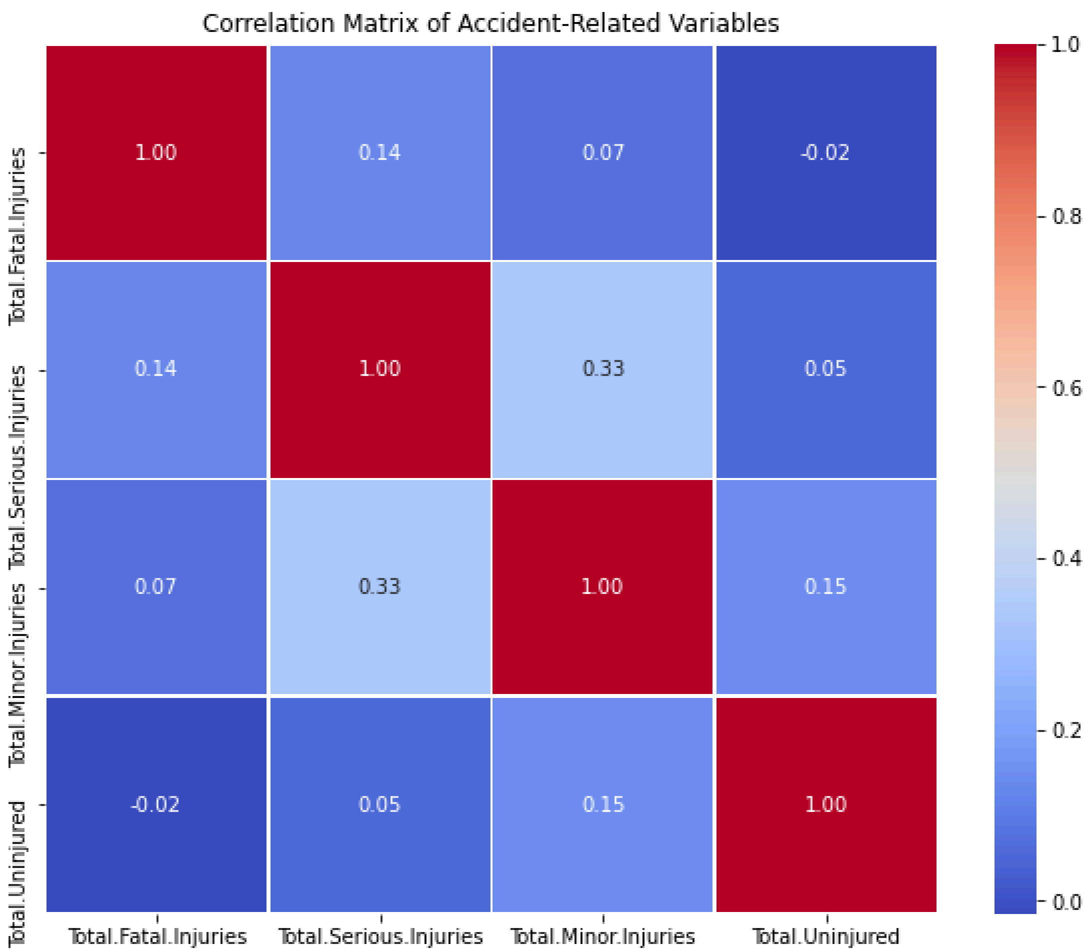
From the line graph the aircraft model with lesser accidents is the "180" We recomend that if the airline is considering, it chooses that as opposed to the aircraft '152'

We now check the accident's correlation

In [ ]:

```
# Select the numerical columns to check correlations (e.g., accident counts, i
numerical_columns = [
    'Total.Fatal.Injuries',
    'Total.Serious.Injuries',
```

```
    'Total.Minor.Injuries','Total.Uninjured']

# Calculate the correlation matrix
correlation_matrix = df[numerical_columns].corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewi
plt.title('Correlation Matrix of Accident-Related Variables')
plt.show()
```



The correlation is mostly negative meaning there is no much correlation between the accident related variables

Look at the location with most accidents to identify regions or routes with higher risk which will help in planning and strategic deployment of the aircraft.

In [ ]:

```
# Check for unique locations to see if there are any data issues
print(df['Location'].unique())
```

```
['MOOSE CREEK, ID' 'BRIDGEPORT, CA' 'Saltville, VA' ... 'San Manual, AZ'
 'Auburn Hills, MI' 'Brasnorte, ']
```

In [74]:

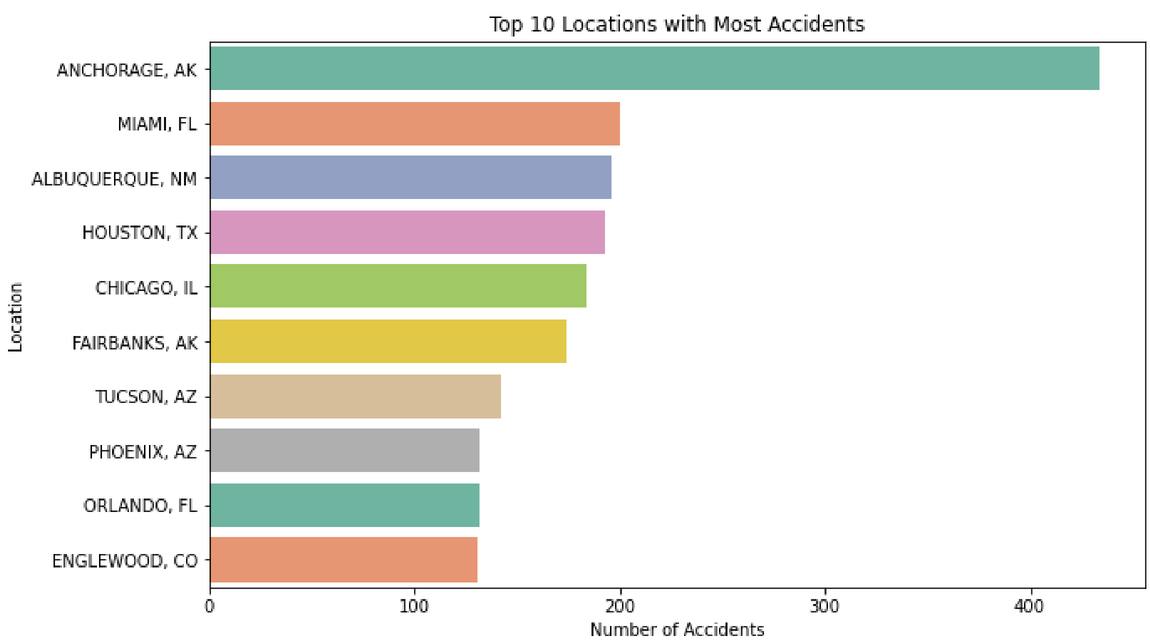
```
# Count the number of accidents by location
```

```
location_accidents = df['Location'].value_counts().reset_index()
location_accidents.columns = ['Location', 'Accident Count']

# Display top locations with the most accidents
print(location_accidents.head(10))
```

```
          Location  Accident Count
0    ANCHORAGE, AK             434
1        MIAMI, FL             200
2  ALBUQUERQUE, NM             196
3      HOUSTON, TX             193
4      CHICAGO, IL             184
5    FAIRBANKS, AK             174
6       TUCSON, AZ             142
7      PHOENIX, AZ             132
8      ORLANDO, FL             132
9    ENGLEWOOD, CO             131
```

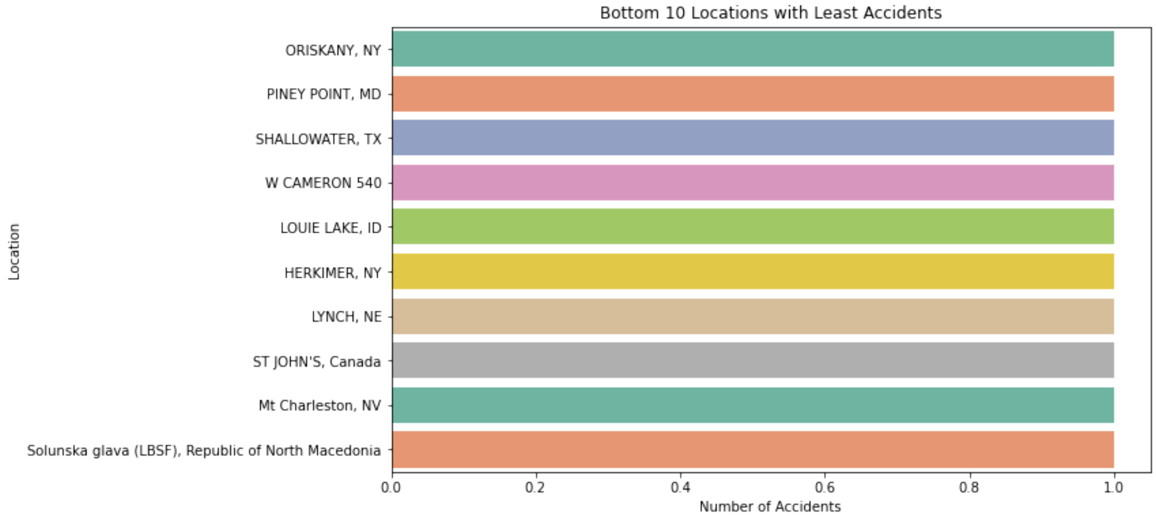
In [75]:

```
# Plotting the top 10 locations with the most accidents
top_locations = location_accidents.head(10)

plt.figure(figsize=(10, 6))
sns.barplot(x='Accident Count', y='Location', data=top_locations, palette='Set
plt.title('Top 10 Locations with Most Accidents')
plt.xlabel('Number of Accidents')
plt.ylabel('Location')
plt.show()
```



In [79]:

```
# Plotting the top 10 locations with the most accidents
bottom_locations = location_accidents.tail(10)

plt.figure(figsize=(10, 6))
sns.barplot(x='Accident Count', y='Location', data=bottom_locations, palette='
plt.title('Bottom 10 Locations with Least Accidents')
plt.xlabel('Number of Accidents')
plt.ylabel('Location')
plt.show()
```



This helped identify accident hotspots and plan for aircraft deployment more effectively, ensuring that regions with higher risk can be monitored or provided with enhanced safety measures. The graph shows the severity in accidents is Anchorage, Ak and the least prone to accidents shown

Conclusion and Recomendations

The analysis indicates that the number of accidents has decreased over time, which could be attributed to improvements in aircraft models, enhanced safety measures in accident-prone areas, and growing experience and expertise.