Final Project Submission

Please fill out:

- Student name: Harrison Kuria Karime
- Student pace: part time
- Scheduled project review date/time: 13/02/25
- Instructor name: Antony Muiko
- Blog post URL:

Data Driven Risk Analysis of Aircraft Safety for Business Expansion

This wlll be a beautiful intro into the projec

Overview

Before taking a deep dive into the available data as our company ventures into this new territory, it is important for us to ask a few important questions. Quesions important to our company and questions that will help us understand whether this new endevour aligns with our future and diretion we mean to take our business. The questions culd look something like this;

- **Aircraft selection**: Which models have the best safety records? Which types are most prone to incidents?
- **Fleet size planning**: How many planes should the company buy initially? Can incident trends inform scaling?
- **Route selection**: Which routes are safest? Are there accident-prone regions?
- **Operational risks**: What common causes lead to accidents? Can the company mitigate these risks?
- **Regulatory compliance**: What aviation regulations are relevant for operational safety?
- **Weather and environmental impact**: How do weather conditions correlate with incidents?

Data Understanding

Getting to have a feel of the data that is provided

In [368…

```
# Your code here - remember to use markdown cells for comments as well!
import pandas as pd
```

```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [369…

```python
# opneing the csv dataset into a DataFrame aviation_data
aviation_data = pd.read_csv('data/Aviation_Data.csv', sep=',', header=0, low_memory
aviation_data.head()
```

Out[369…

	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 [370…

```python
# Getting a general of what the avation dataset looks like
print(f"The shape of aviation_data is :\n", aviation_data.shape)
print()
print(f"Some of the important features of the dataset", aviation_data.info())
print()
print(f"Snapshot of what the numeric data looks like \n", aviation_data.describe())
```

```
The shape of aviation_data is :
 (90348, 31)

<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            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        73659 non-null  object 
dtypes: float64(5), object(26)
memory usage: 21.4+ MB
Some of the important features of the dataset None

Snapshot of what the numeric data looks like
        Number.of.Engines  Total.Fatal.Injuries  Total.Serious.Injuries  \
count       82805.000000          77488.000000            76379.000000   
mean            1.146585              0.647855                0.279881   
std             0.446510              5.485960                1.544084   
min             0.000000              0.000000                0.000000   
25%             1.000000              0.000000                0.000000   
50%             1.000000              0.000000                0.000000   
75%             1.000000              0.000000                0.000000   
max             8.000000            349.000000              161.000000   

       Total.Minor.Injuries  Total.Uninjured  
count          76956.000000     82977.000000  
```

```
mean              0.357061        5.325440
std               2.235625       27.913634
min               0.000000        0.000000
25%               0.000000        0.000000
50%               0.000000        1.000000
75%               0.000000        2.000000
max             380.000000      699.000000
```

Data Cleaning

Identifying Columns with Significant Issues

From the preliminary assessment of the dataset provided by the **NTSB**, we observe that some columns contain significantly more problems than others. This includes **high percentages of missing values** and **inconsistencies in data quality**.

At this stage, it is crucial to:

1. **Decide which columns are not useful for our analysis**—especially those with missing data exceeding 50%.
2. **Identify columns with highly mixed or inconsistent values**, as cleaning them may not be feasible.
3. **Evaluate which columns are relevant to our business questions** to ensure we focus on meaningful insights.

In [373…

```
# Start off by identifying the columns that have a high number of null values
aviation_data.isna().sum()
```

Out[373…

```
Event.Id                   1459
Investigation.Type            0
Accident.Number            1459
Event.Date                 1459
Location                   1511
Country                    1685
Latitude                  55966
Longitude                 55975
Airport.Code              40216
Airport.Name              37644
Injury.Severity            2459
Aircraft.damage            4653
Aircraft.Category         58061
Registration.Number        2841
Make                       1522
Model                      1551
Amateur.Built              1561
Number.of.Engines          7543
Engine.Type                8555
FAR.Description           58325
Schedule                  77766
Purpose.of.flight          7651
Air.carrier               73700
Total.Fatal.Injuries      12860
Total.Serious.Injuries    13969
Total.Minor.Injuries      13392
Total.Uninjured            7371
Weather.Condition          5951
Broad.phase.of.flight     28624
Report.Status              7843
Publication.Date          16689
dtype: int64
```

In [374…

```
np.round(aviation_data.isnull().sum() / len(aviation_data) * 100, 2)
```

Out[374…

```
Event.Id                  1.61
Investigation.Type        0.00
Accident.Number           1.61
Event.Date                1.61
Location                  1.67
Country                   1.87
Latitude                 61.94
Longitude                61.95
Airport.Code             44.51
Airport.Name             41.67
Injury.Severity           2.72
Aircraft.damage           5.15
Aircraft.Category        64.26
Registration.Number       3.14
Make                      1.68
Model                     1.72
Amateur.Built             1.73
Number.of.Engines         8.35
Engine.Type               9.47
FAR.Description          64.56
Schedule                 86.07
Purpose.of.flight         8.47
Air.carrier              81.57
Total.Fatal.Injuries     14.23
Total.Serious.Injuries   15.46
Total.Minor.Injuries     14.82
Total.Uninjured           8.16
Weather.Condition         6.59
Broad.phase.of.flight    31.68
Report.Status             8.68
Publication.Date         18.47
dtype: float64
```

Percentage of Missing Values Per Column

Below is a breakdown of **null value percentages** > 10% in the dataset:

Column Name	Missing Data (%)
Latitude	61.94%
Longitude	61.95%
Airport.Code	44.51%
Airport.Name	41.67%
Aircraft.Category	64.26%
FAR.Description	64.56%
Schedule	86.07%
Air.carrier	81.57%
Broad.phase.of.flight	31.68%

Column Name	Missing Data (%)
Publication.Date	18.47%
Total.Fatal.Injuries	14.23%
Total.Serious.Injuries	15.46%
Total.Minor.Injuries	14.82%

Columns That May Be **Dropped or Prioritized for Cleaning**

- **Columns with Extremely High Missing Data (>50%)**
  - `Latitude` and `Longitude` : Geographical data is missing for over **60%** of the records.
  - `Aircraft.Category` and `FAR.Description` : Missing in **more than 60%** of cases.
  - `Schedule` : Missing in **86%** of records—likely unreliable for analysis.
  - `Air.carrier` : Missing in **81%** of records—may not be useful.

In [376…

```
# Removing the columns with the highest number of missing values
aviation_data.drop(columns=['Latitude', 'Longitude', 'Aircraft.Category', 'FAR.Desc
```

- **Columns That May Not Be Critical to Business Questions**
  - `Registration.Number` : Not needed for general aviation safety or route planning.
  - `Airport.Code` and `Airport.Name` - these have significant number of missing values, and in general may not offer much insight
  - `Publication.Date` : Related to report processing rather than accident causes.

In [378…

```
# Dropping columns with data that is not essential for the business question
aviation_data.drop(columns=['Airport.Code', 'Airport.Name', 'Publication.Date', 'Re
aviation_data.head(2)
```

Out[378…

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Countr
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	Unitec State
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	Unitec State

2 rows × 21 columns

- **FInding event identifing columns that will have use**
- `Event.Id` , `Accident.Number` : These are unique identifiers and **do not contribute to analysis**. However they may be useful be useful further data cleaning or creating useful datasets down the line. Possibly create a dataframe to store them for later use.

There is no need to keep both identifier columns, one may be enough. This will be beneficial down the line especially when dealing with duplicated values.

- `Investigation.Type` - not cruicial for the for the analysis aand goes hand in hand with `Accident.Number`. We may drop both especially if `Investiation.Type` is found to contain alot mixed data types, or does not give us clear picture of what the incidents were like.

In [380…

```python
# Keep one column as unique dentifier of incidents
aviation_data.drop(columns=['Accident.Number', 'Investigation.Type'], inplace=True)
```

In [381…

```python
aviation_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90348 entries, 0 to 90347
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Event.Id                88889 non-null  object 
 1   Event.Date              88889 non-null  object 
 2   Location                88837 non-null  object 
 3   Country                 88663 non-null  object 
 4   Injury.Severity         87889 non-null  object 
 5   Aircraft.damage         85695 non-null  object 
 6   Make                    88826 non-null  object 
 7   Model                   88797 non-null  object 
 8   Amateur.Built           88787 non-null  object 
 9   Number.of.Engines       82805 non-null  float64
 10  Engine.Type             81793 non-null  object 
 11  Purpose.of.flight       82697 non-null  object 
 12  Total.Fatal.Injuries    77488 non-null  float64
 13  Total.Serious.Injuries  76379 non-null  float64
 14  Total.Minor.Injuries    76956 non-null  float64
 15  Total.Uninjured         82977 non-null  float64
 16  Weather.Condition       84397 non-null  object 
 17  Broad.phase.of.flight   61724 non-null  object 
 18  Report.Status           82505 non-null  object 
dtypes: float64(5), object(14)
memory usage: 13.1+ MB
```

In [382…

```python
aviation_data.isna().sum()
```

Out[382…
```
Event.Id                    1459
Event.Date                  1459
Location                    1511
Country                     1685
Injury.Severity             2459
Aircraft.damage             4653
Make                        1522
Model                       1551
Amateur.Built               1561
Number.of.Engines           7543
Engine.Type                 8555
Purpose.of.flight           7651
Total.Fatal.Injuries       12860
Total.Serious.Injuries     13969
Total.Minor.Injuries       13392
Total.Uninjured             7371
Weather.Condition           5951
Broad.phase.of.flight      28624
Report.Status               7843
dtype: int64
```

In [383…
```python
print(f"Counts based on state of plane at crash site:\n", aviation_data['Aircraft.d
```

```
Counts based on state of plane at crash site:
 Aircraft.damage
Substantial    64148
Destroyed      18623
Minor           2805
Unknown          119
Name: count, dtype: int64
```

Next Steps

1. **Impute or clean columns with moderate missing data and inconsitencies** if they are critical to the analysis.
2. **Focus on columns that answer key business questions**, such as:
   - **Aircraft safety**: `Make`, `Model`, `Number.of.Engines`, `Engine.Type`, `'Aircraft.damage`
   - **Route risk assessment**: `Location`, `Weather.Condition`
   - **Operational risks**: `Broad.phase.of.flight`, `Purpose.of.flight`
   - **Human safety impact**: `Injury.Severity`, `Total.Fatal.Injuries`

Step 1: Fixing data inconstencies

- Standardized text fields by converting them to **uppercase** and **removing extra spaces**.
- Ensured consistency in categorical values like `Country`, `Make`, and `Weather.Condition` to **prevent duplicates due to case differences**.

In [386…
```python
# creating a listof columns containing text
text_col = ['Location', 'Country', 'Injury.Severity', 'Aircraft.damage', 'Aircraft.
```

```
            'Purpose.of.flight', 'Weather.Condition', 'Broad.phase.of.flight', 'Rep

# Dealing with white spaces and trailing characters
for column in text_col:
    aviation_data[column] = aviation_data[column].astype(str).str.strip().str.upper

aviation_data.sample(5)
```

Out[386…

	Event.Id	Event.Date	Location	Country	Injury.Severity	Aircraft.dama
15567	20001213X34521	1986-08-09	RANCHO MIRAGE, CA	UNITED STATES	NON-FATAL	SUBSTANTI
17573	20001213X30861	1987-05-22	HUNTERSVILLE, NC	UNITED STATES	NON-FATAL	SUBSTANTI
10105	20001214X41412	1984-10-20	APPLE VALLEY, CA	UNITED STATES	NON-FATAL	SUBSTANTI
60604	20060607X00694	2006-05-07	WEST MIDDLESEX, PA	UNITED STATES	NON-FATAL	SUBSTANTI
39392	20001208X05629	1996-04-14	KISSIMMEE, FL	UNITED STATES	FATAL(1)	DESTROY

In [387…

```
aviation_data['Country'].value_counts()
```

Out[387…

```
Country
UNITED STATES    82248
NAN               1685
BRAZIL             374
CANADA             359
MEXICO             358
                 ...
MAURITANIA           1
OBYAN                1
WOLSELEY             1
ALBANIA              1
GUERNSEY             1
Name: count, Length: 216, dtype: int64
```

Step 2: Handling Duplicates

- Identified potential **duplicate records** using `Event.Id`, `Event.Date`, `Make`, and `Model`.
- Remove **exact duplicate records** to maintain dataset integrity.

> Taking into account that there are columns that we anticipate duplicates, we want to focus on unique identifiers, e.g. `Event.Id`, `Event.Date`

In [389...

```
# inspect duplicated rows
print(f"The sum of exact duplicated rows is: ", aviation_data.duplicated().sum())
print()
print('Below a sample of duplicate rows')
aviation_data[aviation_data.duplicated(keep=False)]
```

```
The sum of exact duplicated rows is:  1486

Below a sample of duplicate rows
```

Out[389...

	Event.Id	Event.Date	Location	Country	Injury.Severity	Aircraft.damage
1370	20020917X02935	1982-05-28	EVANSVILLE, IN	UNITED STATES	NON-FATAL	SUBSTANTIAL
1371	20020917X02935	1982-05-28	EVANSVILLE, IN	UNITED STATES	NON-FATAL	SUBSTANTIAL
3081	20020917X04638	1982-10-18	GULF OF MEXICO	GULF OF MEXICO	FATAL(3)	DESTROYED
3082	20020917X04638	1982-10-18	GULF OF MEXICO	GULF OF MEXICO	FATAL(3)	DESTROYED
4760	20001214X43016	1983-05-22	BRIDGEPORT, CA	UNITED STATES	FATAL(1)	SUBSTANTIAL
...	...	...	...	...	...	...
90004	NaN	NaN	NAN	NAN	NAN	NAN
90010	NaN	NaN	NAN	NAN	NAN	NAN
90031	NaN	NaN	NAN	NAN	NAN	NAN
90090	NaN	NaN	NAN	NAN	NAN	NAN
90097	NaN	NaN	NAN	NAN	NAN	NAN

1515 rows × 19 columns

> From the sampling above the duplicated rows generally seem to be exact copies or containing NaN values, so the best move is to drop and only keep one of the copied rows.

In [391...

```
# Now dropping the duplicated rows
aviation_data = aviation_data.drop_duplicates(subset='Event.Id', keep='first')
aviation_data
```

Out[391…

	Event.Id	Event.Date	Location	Country	Injury.Severity	Aircraft.damage
0	20001218X45444	1948-10-24	MOOSE CREEK, ID	UNITED STATES	FATAL(2)	DESTROYED
1	20001218X45447	1962-07-19	BRIDGEPORT, CA	UNITED STATES	FATAL(4)	DESTROYED
2	20061025X01555	1974-08-30	SALTVILLE, VA	UNITED STATES	FATAL(3)	DESTROYED
3	20001218X45448	1977-06-19	EUREKA, CA	UNITED STATES	FATAL(2)	DESTROYED
4	20041105X01764	1979-08-02	CANTON, OH	UNITED STATES	FATAL(1)	DESTROYED
...	...	...	...	...	...	...
90343	20221227106491	2022-12-26	ANNAPOLIS, MD	UNITED STATES	MINOR	NAN
90344	20221227106494	2022-12-26	HAMPTON, NH	UNITED STATES	NAN	NAN
90345	20221227106497	2022-12-26	PAYSON, AZ	UNITED STATES	NON-FATAL	SUBSTANTIAL
90346	20221227106498	2022-12-26	MORGAN, UT	UNITED STATES	NAN	NAN
90347	20221230106513	2022-12-29	ATHENS, GA	UNITED STATES	MINOR	NAN

87952 rows × 19 columns

Step 3: Handling Missing Values

- **Filled missing numeric values** (e.g., `Total.Fatal.Injuries`) with `0` to avoid misinterpretation.
- **Imputed categorical fields** (`Engine.Type`, `Weather.Condition`, `Broad.phase.of.flight`) with **mode or "UNKNOWN"** to retain important data.
- Ensured missing values in **critical analysis columns** were handled appropriately without losing essential insights.

> Here the idea is we will go column by column assessing the folders with missing values and implement at one of the approaches above

In [393…

```
# Checking for columns still with missing values
aviation_data.isna().sum()
```

Out[393…

```
Event.Id                    1
Event.Date                  1
Location                    0
Country                     0
Injury.Severity             0
Aircraft.damage             0
Make                        0
Model                       0
Amateur.Built               0
Number.of.Engines        6028
Engine.Type                 0
Purpose.of.flight           0
Total.Fatal.Injuries    11268
Total.Serious.Injuries  12323
Total.Minor.Injuries    11761
Total.Uninjured          5864
Weather.Condition           0
Broad.phase.of.flight       0
Report.Status               0
dtype: int64
```

Understanding the Engine Counts Column

- Every aircraft must have at least **one engine or propulsion system**, whether traditional or electric.
- The `Number.of.Engines` column should **not have missing values** since all operational aircraft require propulsion.

Identifying Missing and Unusual Values To ensure consistency, we first check for:

- **Missing values** in the `Number.of.Engines` column.
- **Unusual values** that may indicate data entry errors. For example `0` engines on a plane

> By analysing this columns statistics we can establish the most common type of engine count and assume these planes with **NaN** will likely have that too, or inspect by model and whether that make sense

In [395…

```
# checking through the column Number.of.engines
print(f"The number of rows missing engine count: ", aviation_data['Number.of.Engine
print()
print(f"The distribution of engine counts looks like this: ", aviation_data['Number
print()

# finding the most common engine number
most_common_count = aviation_data['Number.of.Engines'].mode()[0]
print(f"The most common number of engines is", most_common_count)
```

```
The number of rows missing engine count:  6028

The distribution of engine counts looks like this:  Number.of.Engines
1.0    68956
2.0    10891
NaN     6028
0.0     1210
3.0      448
4.0      415
8.0        3
6.0        1
Name: count, dtype: int64

The most common number of engines is 1.0
```

In [396…

```
# inspecting the row with zero and whether they are data entry errors in general
aviation_data[aviation_data['Number.of.Engines'] == 0].sample(5)
```

Out[396…

	Event.Id	Event.Date	Location	Country	Injury.Severity	Aircraft.damag
33232	20001211X12951	1993-07-12	LA JOLLA, CA	UNITED STATES	NON-FATAL	NA
18608	20001213X32059	1987-09-05	COLORADO SPRING, CO	UNITED STATES	NON-FATAL	SUBSTANTI
32655	20001211X12388	1993-05-01	BENTONVILLE, AR	UNITED STATES	NON-FATAL	SUBSTANTI
63240	20070908X01331	2007-08-25	HEBER, UT	UNITED STATES	NON-FATAL	SUBSTANTI
30705	20001211X14982	1992-06-23	WAYNESBORO, VA	UNITED STATES	NON-FATAL	SUBSTANTI

In [397…

```
# replacing the null values with the mode
aviation_data.loc[:, 'Number.of.Engines'] = aviation_data['Number.of.Engines'].fill
```

In [398…

```
# As for the zero engine count, replace '0' with mode as well
# It may be tideous to identfy each plane by make and model, these columns are way
aviation_data.loc[:, 'Number.of.Engines'] = aviation_data['Number.of.Engines'].repl
```

In [399…

```
# Checking to see if the column has been fixed
print(f"The Number of NaN values is", aviation_data['Number.of.Engines'].isna().sum
print(f"What the distribution of engine counts looks like this after fixing NaN and
```

```
The Number of NaN values is 0
What the distribution of engine counts looks like this after fixing NaN and zero val
ues:  Number.of.Engines
1.0    76194
2.0    10891
3.0      448
4.0      415
8.0        3
6.0        1
Name: count, dtype: int64
```

Cleaning and Validating The Injury Report Columns

1. Logical Approach to Handling Missing and Inconsistent Values Unlike other dataset fields, the **report columns** contain **recorded deaths and injuries**, which require **careful handling** to avoid **inaccuracies**.
Simply replacing missing values with the **mode or mean** is **not appropriate** because:

- **Deaths and injuries should always be documented accurately**.
- **Negative values** are **impossible** and must be removed.
- **Missing (`NaN`) values** must be handled based on **logical assumptions** rather than standard imputation.

---

2. Handling Missing Values in Report Columns To maintain accuracy:

- `Total.Fatal.Injuries` → If `NaN`, assume **0**, since fatalities would have been reported.
- `Total.Serious.Injuries` & `Total.Minor.Injuries` → If `NaN`, assume **0**, as serious and minor injuries are usually documented.
- `Total.Uninjured` → If `NaN`, assume **0**, as uninjured passengers may not be explicitly recorded.

In [401…

```python
# create a list of injury report columns
injury_report_columns =['Total.Fatal.Injuries', 'Total.Serious.Injuries', 'Total.Mi

# replacing the NaN with zero
aviation_data.loc[:, injury_report_columns] = aviation_data[injury_report_columns].

# replacing negative values with zero
for col in injury_report_columns:
    aviation_data[col] = aviation_data[col].apply(lambda x: 0 if x<0 else x)

# validating if NaN values have been fixed
print(aviation_data[injury_report_columns].isna().sum())
```

```
Total.Fatal.Injuries      0
Total.Serious.Injuries    0
Total.Minor.Injuries      0
Total.Uninjured           0
dtype: int64
```

In [402…
```
# spotting that something is off with the injury severity column
aviation_data['Injury.Severity']
```

Out[402…
```
0          FATAL(2)
1          FATAL(4)
2          FATAL(3)
3          FATAL(2)
4          FATAL(1)
             ...
90343         MINOR
90344           NAN
90345     NON-FATAL
90346           NAN
90347         MINOR
Name: Injury.Severity, Length: 87952, dtype: object
```

In [403…
```
# we will deploy a formula to standadize the injury report column, that with will h
import re

def std_injury_severity(col_value):
    col_value = str(col_value).strip(col_value).upper()
    # converting all variations Fata(x) to fatal
    if "FATAL" in col_value:
        return "Fatal"
    elif col_value in ["NON-FATAL", "MINOR", "SERIOUS", "INCIDENT"]:
        return col_value
    else:
        return "unknown"

# applying the formula to the dataset
aviation_data['Injury.Severity'].apply(std_injury_severity)
```

Out[403…
```
0        unknown
1        unknown
2        unknown
3        unknown
4        unknown
          ...
90343    unknown
90344    unknown
90345    unknown
90346    unknown
90347    unknown
Name: Injury.Severity, Length: 87952, dtype: object
```

In [404…
```
# Convert Event.Date to datetime format
aviation_data['Event.Date'] = pd.to_datetime(aviation_data['Event.Date'], errors='c
```

In [405…
```
aviation_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 87952 entries, 0 to 90347
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Event.Id                87951 non-null  object
 1   Event.Date              87951 non-null  datetime64[ns]
 2   Location                87952 non-null  object
 3   Country                 87952 non-null  object
 4   Injury.Severity         87952 non-null  object
 5   Aircraft.damage         87952 non-null  object
 6   Make                    87952 non-null  object
 7   Model                   87952 non-null  object
 8   Amateur.Built           87952 non-null  object
 9   Number.of.Engines       87952 non-null  float64
 10  Engine.Type             87952 non-null  object
 11  Purpose.of.flight       87952 non-null  object
 12  Total.Fatal.Injuries    87952 non-null  float64
 13  Total.Serious.Injuries  87952 non-null  float64
 14  Total.Minor.Injuries    87952 non-null  float64
 15  Total.Uninjured         87952 non-null  float64
 16  Weather.Condition       87952 non-null  object
 17  Broad.phase.of.flight   87952 non-null  object
 18  Report.Status           87952 non-null  object
dtypes: datetime64[ns](1), float64(5), object(13)
memory usage: 13.4+ MB
```

In [406…

```
# Sampling through our cleaned dataset
aviation_data.sample(10)
```

Out[406…

	Event.Id	Event.Date	Location	Country	Injury.Severity	Aircraft.damag
75734	20140718X92314	2014-07-17	HRABOVE, UKRAINE	UKRAINE	FATAL	DESTROYE
8592	20001214X40136	1984-06-12	PASCO, WA	UNITED STATES	NON-FATAL	SUBSTANTIA
25325	20001212X22870	1990-04-22	SANDWICH, IL	UNITED STATES	NON-FATAL	SUBSTANTIA
71228	20111214X31105	2011-12-14	TUCSON, AZ	UNITED STATES	NON-FATAL	SUBSTANTIA
20549	20001213X25958	1988-06-13	MOUNDS, OK	UNITED STATES	NON-FATAL	SUBSTANTIA
33919	20001211X13524	1993-10-02	ROODHOUSE, IL	UNITED STATES	FATAL(2)	DESTROYE
86029	20200709X11322	2020-07-04	OOLTEWAH, TN	UNITED STATES	NON-FATAL	SUBSTANTIA
89459	20220627105370	2022-06-27	MOAB, UT	UNITED STATES	NON-FATAL	SUBSTANTIA
83866	20190329X55659	2019-03-29	ORMOND BEACH, FL	UNITED STATES	NON-FATAL	SUBSTANTIA
58735	20050615X00771	2005-05-29	OKLAHOMA CITY, OK	UNITED STATES	NON-FATAL	SUBSTANTIA

Handling String `"NaN"` Values in Categorical Data

Identifying the Issue

During manual sampling of the cleaned dataset, we notice an inconsistency:
Some categorical column values were stored as **the string `"NaN"`** instead of the standard `np.nan` (Python's default missing value representation).

Because these values were **not recognized as actual missing values (`NaN`)**, they were **not handled properly** during the initial data cleaning phase.

Converting `"NaN"` Strings to Actual `NaN`

To ensure proper handling of missing data, we replace all occurrences of `"NaN"` (as a string) with **`np.nan`**. Then the option here is to replace them with unknown, this would seem like a fitting category in the different categorical column as opposed to dropping the values altogether.

In [408…
```
# Replace string "NaN" with actual NaN values
aviation_data.replace("NAN", "UNKNOWN", inplace=True)
aviation_data.isna().sum()
```

Out[408…
```
Event.Id                  1
Event.Date                1
Location                  0
Country                   0
Injury.Severity           0
Aircraft.damage           0
Make                      0
Model                     0
Amateur.Built             0
Number.of.Engines         0
Engine.Type               0
Purpose.of.flight         0
Total.Fatal.Injuries      0
Total.Serious.Injuries    0
Total.Minor.Injuries      0
Total.Uninjured           0
Weather.Condition         0
Broad.phase.of.flight     0
Report.Status             0
dtype: int64
```

In [409…
```
aviation_data.isna().sum()
```

Out[409…

```
Event.Id                  1
Event.Date                1
Location                  0
Country                   0
Injury.Severity           0
Aircraft.damage           0
Make                      0
Model                     0
Amateur.Built             0
Number.of.Engines         0
Engine.Type               0
Purpose.of.flight         0
Total.Fatal.Injuries      0
Total.Serious.Injuries    0
Total.Minor.Injuries      0
Total.Uninjured           0
Weather.Condition         0
Broad.phase.of.flight     0
Report.Status             0
dtype: int64
```

In [410…

```
# writng our cleaned dataset into a csv file
aviation_data.to_csv('data/cleaned_aviation_data.csv', index=True)
```

Step 4: Generating Meaningful Data Subsets

> Finally with our data clean enough we can move creating meaningful datasets

Aircraft Safety Analysis

- Grouped aircraft by **Make & Model**, calculating:
  - **Total accidents per aircraft model**
  - **Total fatalities**
  - **Fatality rate (fatalities per accident)**
- Sorted aircraft to determine **which models are safest vs. most accident-prone**.

In [413…

```
# using groupby() to create an aggregated DataFame of aircraft with the human toll
aicraft_safety_df = aviation_data.groupby(['Make', 'Model']).agg(
    total_crashes = ('Event.Id', 'count'),
    total_fatalities = ('Total.Fatal.Injuries', 'sum'),
    total_serious_injuries = ('Total.Serious.Injuries', 'sum'),
    total_minor_injurie = ('Total.Minor.Injuries', 'sum')
).reset_index()
```

In [414…

```
# Top 10 aircraft makes associated with highest fatality
aicraft_safety_df.sort_values('total_fatalities', ascending=False).head(10)
```

Out[414...

	Make	Model	total_crashes	total_fatalities	total_serious_injuries	total_minor_inj
3153	BOEING	737	484	1348.0	388.0	
3189	BOEING	737-200	51	906.0	88.0	
3437	BOEING	777 - 206	3	534.0	0.0	
3587	BOEING	MD-82	8	403.0	2.0	
4650	CESSNA	172N	1143	402.0	201.0	3
4599	CESSNA	172	1740	386.0	310.0	3
841	AIRBUS	A321	20	381.0	0.0	
13341	PIPER	PA-28-181	520	377.0	112.0	1
4575	CESSNA	152	2312	351.0	196.0	4
17087	TUPOLEV	TU-154	1	349.0	0.0	

Getting aircraft fatality rate ie. Which air crafts have higher total fatalities, event when their total count of incedents is compaatively lower

In [416...

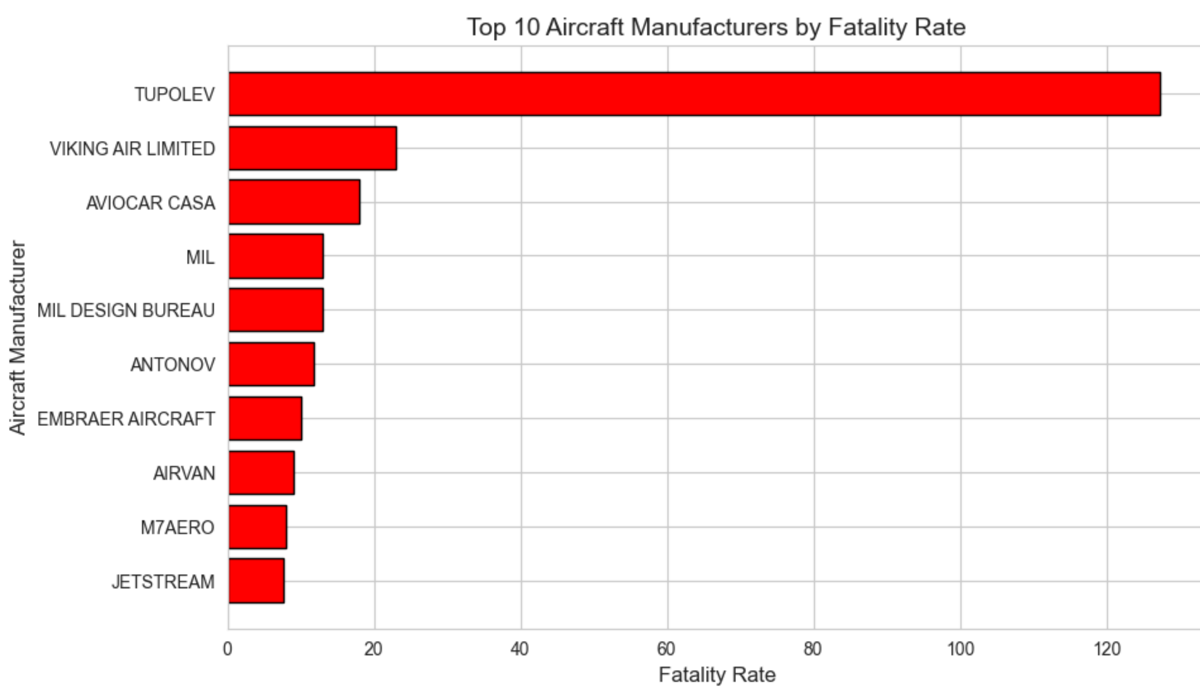
```
# aggregate the columns to help with the calculation
fatality_rate_df = aviation_data.groupby('Make').agg(
    total_crashes = ('Event.Id', 'count'),
     total_fatalities = ('Total.Fatal.Injuries', 'sum'),
).reset_index()

# fatality rate calculated as total fatal injuries of a particular make divided the
fatality_rate_df['Fatality.Rates'] = np.round(fatality_rate_df['total_fatalities']

fatality_rate_df = fatality_rate_df.sort_values(by='Fatality.Rates', ascending=Fals
fatality_rate_df
```

Out[416...

	Make	total_crashes	total_fatalities	Fatality.Rates
6931	TUPOLEV	4	509.0	127.25
7104	VIKING AIR LIMITED	1	23.0	23.00
470	AVIOCAR CASA	1	18.0	18.00
4639	MIL	1	13.0	13.00
4640	MIL DESIGN BUREAU	1	13.0	13.00
345	ANTONOV	6	71.0	11.83
2144	EMBRAER AIRCRAFT	1	10.0	10.00
208	AIRVAN	1	9.0	9.00
4263	M7AERO	1	8.0	8.00
3563	JETSTREAM	3	23.0	7.67

In [417...

```
# getting of the above top 10 riskiest makes determined determined by few crashes c
# Plot
plt.figure(figsize=(10, 6))
plt.barh(fatality_rate_df['Make'], fatality_rate_df['Fatality.Rates'], color='red',

# Labels and title
plt.xlabel("Fatality Rate", fontsize=12)
plt.ylabel("Aircraft Manufacturer", fontsize=12)
plt.title("Top 10 Aircraft Manufacturers by Fatality Rate", fontsize=14)
plt.gca().invert_yaxis()  # Invert to show highest fatality rate on top

# Show plot
plt.show()
```



In [418…
```
fatality_rate_df.to_csv('data/fatality_rate.csv', index=True)
```

Route Risk Analysis

- Counted **total incidents per country** to identify **high-risk regions**.
- Helps inform **route selection for safety and operational planning**.

In [420…
```
route_risk_df = aviation_data.groupby('Country')['Event.Id'].count().reset_index()
route_risk_df.columns = ('Country', 'Total.Accidents')

# top riskies and top safes routes to take
top_20_risk = route_risk_df.sort_values(by='Total.Accidents', ascending=False).head
top_20_risk
```

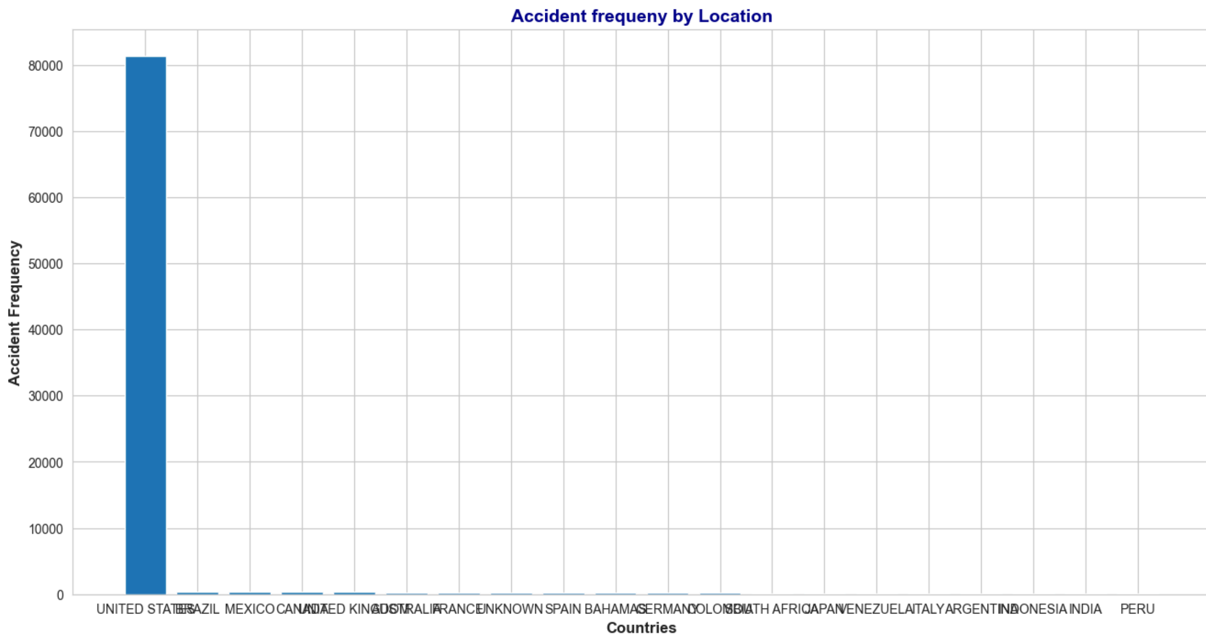
Out[420…

	Country	Total.Accidents
203	UNITED STATES	81355
27	BRAZIL	373
123	MEXICO	356
32	CANADA	355
202	UNITED KINGDOM	341
11	AUSTRALIA	300
64	FRANCE	235
204	UNKNOWN	225
181	SPAIN	224
14	BAHAMAS	215
70	GERMANY	210
41	COLOMBIA	193
177	SOUTH AFRICA	129
99	JAPAN	125
206	VENEZUELA	121
96	ITALY	113
8	ARGENTINA	111
90	INDONESIA	110
89	INDIA	94
150	PERU	93

In [421…
```
top_20_risk.to_csv('data/top_20_hazard_route.csv', index=True)
```

In [422…
```
# plot of the top 20 riskiest routes in terms of incident numbers
fig, ax = plt.subplots(figsize=(16, 8))

ax.bar(top_20_risk['Country'], top_20_risk['Total.Accidents'])

ax.set_title("Accident frequeny by Location", fontsize=14, fontweight='bold', color
ax.set_xlabel("Countries", fontsize=12, fontweight='bold')
ax.set_ylabel("Accident Frequency", fontsize=12, fontweight='bold')
```

Out[422…
```
Text(0, 0.5, 'Accident Frequency')
```



Operational Risk Analysis

- Aggregated **accidents by flight phase** (e.g., Takeoff, Landing, Cruise).
- Helps pinpoint **which stages of flight are most dangerous**.
- Aloso we can consider **risk based on purpose** e.g. hyothesis ,Pilots in training or those in races are more prone to bad events

Accidents by flight phase

In [425…
```
# accident risk based on the broad phases of flights
operational_risk_df = aviation_data.groupby('Broad.phase.of.flight')['Event.Id'].co
operational_risk_df
```

Out[425…

	Broad.phase.of.flight	Event.Id
0	APPROACH	6389
1	CLIMB	1995
2	CRUISE	10141
3	DESCENT	1870
4	GO-AROUND	1345
5	LANDING	15320
6	MANEUVERING	8052
7	OTHER	116
8	STANDING	872
9	TAKEOFF	12404
10	TAXI	1786
11	UNKNOWN	27661

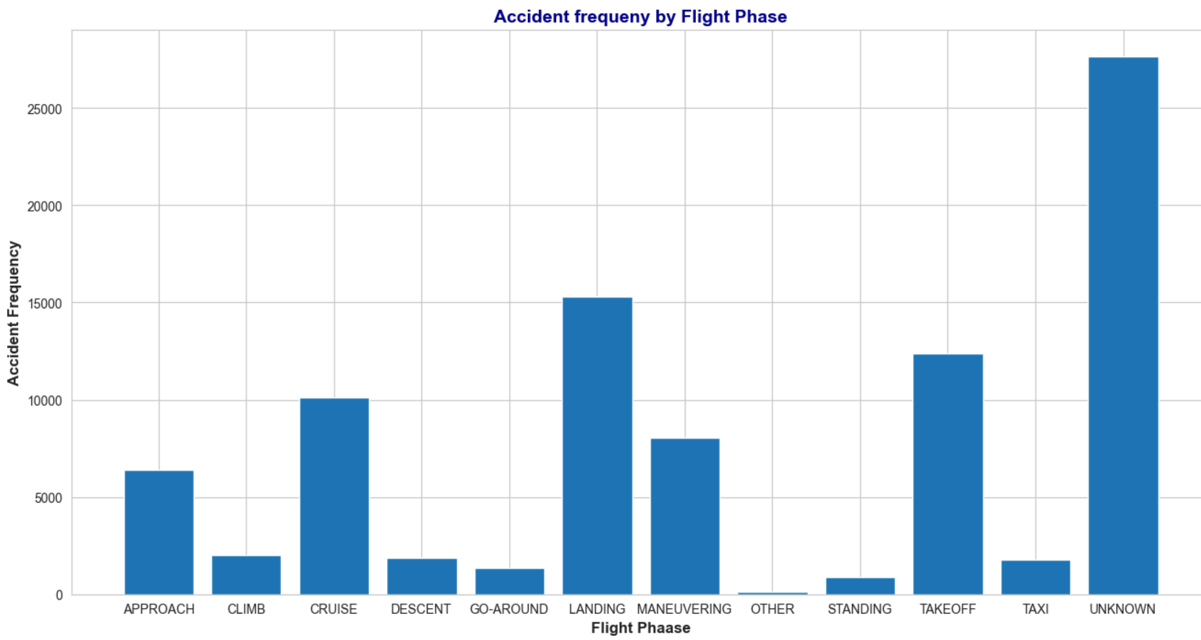
A plot showing during which phases of flight an accident is likely to ocur

In [427…

```
# A plot showing whih phase of flight anncident is mot likeley to occur
sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(16, 8))

ax.bar(operational_risk_df['Broad.phase.of.flight'], operational_risk_df['Event.Id'

ax.set_title("Accident frequeny by Flight Phase", fontsize=14, fontweight='bold', c
ax.set_xlabel("Flight Phaase", fontsize=12, fontweight='bold')
ax.set_ylabel("Accident Frequency", fontsize=12, fontweight='bold')
```

Out[427…

```
Text(0, 0.5, 'Accident Frequency')
```



> The unknown column is significantly taller this may imply that more thorough investigation or factors not considered may be needed.

Accidents based on Purpose of Fligh

In [430…

```
# based on the purpose of the flight
flight_purpose_df = aviation_data.groupby('Purpose.of.flight')['Event.Id'].count().
flight_purpose_df.sort_values(by='Event.Id', ascending=False)
```

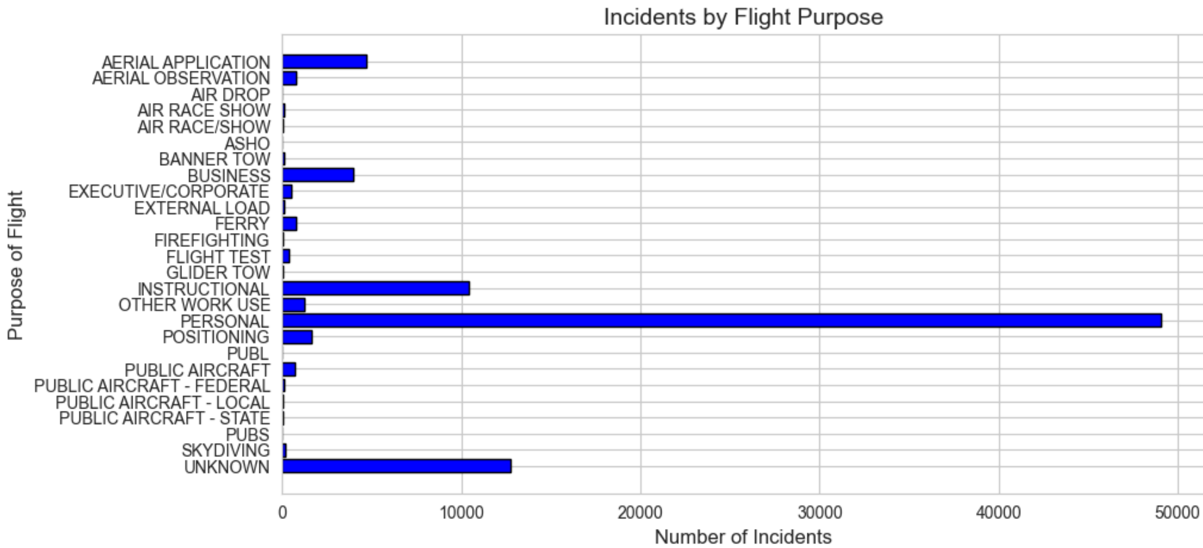
Out[430…

	Purpose.of.flight	Event.Id
16	PERSONAL	49076
25	UNKNOWN	12731
14	INSTRUCTIONAL	10442
0	AERIAL APPLICATION	4686
7	BUSINESS	3971
17	POSITIONING	1632
15	OTHER WORK USE	1250
10	FERRY	806
1	AERIAL OBSERVATION	787
19	PUBLIC AIRCRAFT	710
8	EXECUTIVE/CORPORATE	542
12	FLIGHT TEST	405
24	SKYDIVING	181
9	EXTERNAL LOAD	123
20	PUBLIC AIRCRAFT - FEDERAL	104
6	BANNER TOW	101
3	AIR RACE SHOW	99
21	PUBLIC AIRCRAFT - LOCAL	74
22	PUBLIC AIRCRAFT - STATE	64
13	GLIDER TOW	53
4	AIR RACE/SHOW	53
11	FIREFIGHTING	40
2	AIR DROP	11
5	ASHO	5
23	PUBS	4
18	PUBL	1

In [431…

```
# Plot
plt.figure(figsize=(10, 5))
plt.barh(flight_purpose_df['Purpose.of.flight'], flight_purpose_df['Event.Id'], col

# Labels and title
plt.xlabel("Number of Incidents", fontsize=12)
plt.ylabel("Purpose of Flight", fontsize=12)
```

```
plt.title("Incidents by Flight Purpose", fontsize=14)
plt.gca().invert_yaxis()  # Puts highest category on top

# Show plot
plt.show()
```



> Personal flights have way more incedets than the rest, this could be due to a larger proportions of amateur pilots with smaller planes lacking less automation, and other aides. Plus less training and not using air traffick control adequately

Weather and Environmental Impact

- Analyzed **weather conditions linked to accidents**.
- Helps evaluate how **weather impacts aviation safety** and influences **flight scheduling**.

In [434…
```
aviation_data['Weather.Condition'].value_counts()
```

Out[434…
```
Weather.Condition
VMC        76417
IMC         5949
UNKNOWN     4474
UNK         1112
Name: count, dtype: int64
```

In [435…
```
weather_impact_df = aviation_data.groupby('Weather.Condition')['Event.Id'].count().
weather_impact_df.columns = ['Weather Condition', 'Total Accidents']
weather_impact_df
```

Out[435…

	Weather Condition	Total Accidents
0	IMC	5949
1	UNK	1112
2	UNKNOWN	4473
3	VMC	76417

Accident trends over time

The anticipation here is that as technology has advanced this has inversley affected the chances of incidents in the air

In [437…

```
# creating DF to show these trends
aviation_data['Year'] = aviation_data['Event.Date'].dt.year
accident_trend_df = aviation_data.groupby('Year')['Event.Id'].count().reset_index()
accident_trend_df.columns = ['Year', 'Total Accidents']
accident_trend_df
```

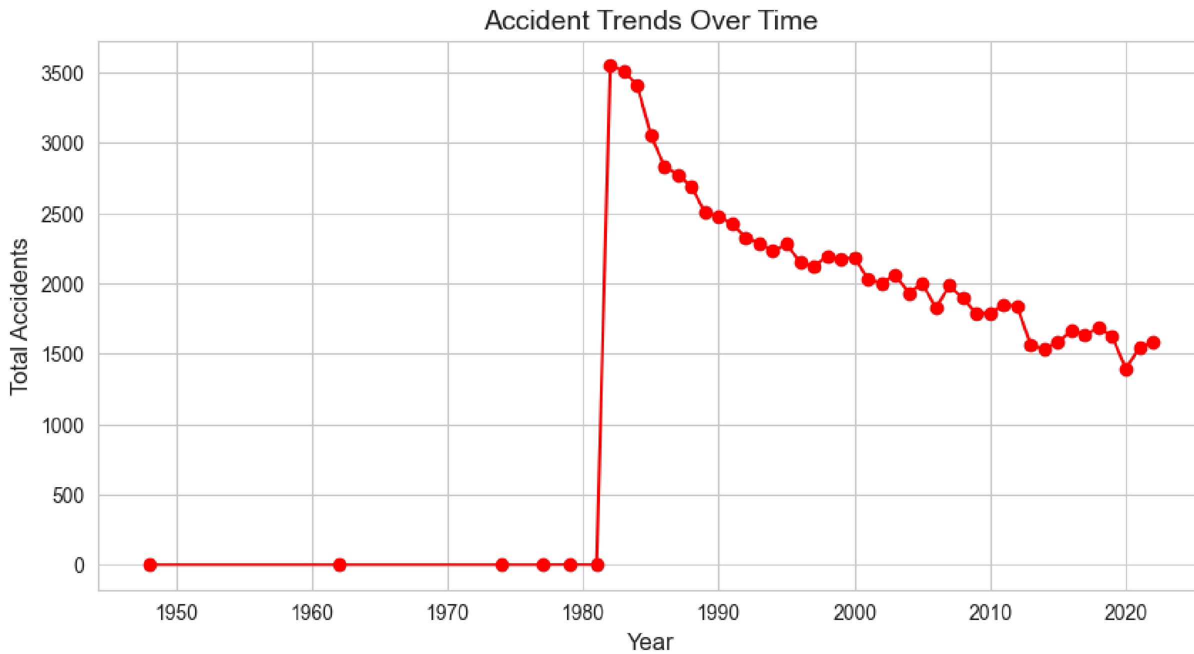
Out[437…

	Year	Total Accidents
0	1948.0	1
1	1962.0	1
2	1974.0	1
3	1977.0	1
4	1979.0	2
5	1981.0	1
6	1982.0	3547
7	1983.0	3513
8	1984.0	3406
9	1985.0	3053
10	1986.0	2832
11	1987.0	2773
12	1988.0	2685
13	1989.0	2502
14	1990.0	2480
15	1991.0	2420
16	1992.0	2328
17	1993.0	2285
18	1994.0	2229
19	1995.0	2278
20	1996.0	2150
21	1997.0	2121
22	1998.0	2196
23	1999.0	2174
24	2000.0	2183
25	2001.0	2032
26	2002.0	2001
27	2003.0	2063
28	2004.0	1932
29	2005.0	2001

	Year	Total Accidents
30	2006.0	1826
31	2007.0	1984
32	2008.0	1893
33	2009.0	1783
34	2010.0	1786
35	2011.0	1850
36	2012.0	1835
37	2013.0	1561
38	2014.0	1535
39	2015.0	1582
40	2016.0	1664
41	2017.0	1638
42	2018.0	1681
43	2019.0	1624
44	2020.0	1392
45	2021.0	1545
46	2022.0	1581

In [438…

```
# Plot accident trends over time using a line graph
plt.figure(figsize=(10, 5))
plt.plot(accident_trend_df['Year'], accident_trend_df['Total Accidents'], marker='o

# Labels and title
plt.xlabel("Year", fontsize=12)
plt.ylabel("Total Accidents", fontsize=12)
plt.title("Accident Trends Over Time", fontsize=14)

# Show grid for better readability
plt.grid(True)
```



> Things seem relatively quiet in the erlier years but this is probably due to less flights but also maybe because of less reporting and records. The spike in the 80s could be due to a inrease in flights and also established of regulatory bodies and more investigation leading to beetter records. But all in all there seems be a decrease in incidents, better technology, more regulations and safeguards

---

Key Business Insights From Analysis

Overview

This report provides **data-driven insights** from a comprehensive analysis of aviation accident data.
It highlights **key risk factors**, **aircraft safety trends**, **route risks**, **operational challenges**, and **weather impacts** that are **critical** for a company considering entry into the aviation sector.

Key Insights from the Data

1. Aircraft Selection – Which Models Have the Best Safety Records?

- The **most accident-prone aircraft models** include:
  - **Tupolev TU-154** – High fatality rate due to severe accidents.
  - **Boeing 777-206** – Large aircraft involved in multiple serious incidents.
  - **McDonnell Douglas DC-8-62** – Several incidents with high fatalities.
- The **safest aircraft models** tend to be those with **modern technology, strong safety records, and low accident frequencies**.

Insight

- Invest in **modern aircraft models** with **strong safety records** and **lower accident rates**.
- Prioritize **robust builds, proven reliability, and lower maintenance costs**.
- **Lease newer aircraft** before buying

---

2. Fleet Planning – How Many Planes Should the Company Buy Initially?

- **Accident trends have declined over time**, showing **improvements in aviation safety**.
- **data-driven approach** to **fleet size planning** should consider:
  - **Historical accident rates per aircraft model**.
  - **Passenger demand & revenue models**.
  - **Route expansion strategies**.

Insight:

- Start with **a small fleet** to test market demand and ensure **operational efficiency**.
- Grow the fleet as the aviation industry becomes more common place for the company

---

3. Route Selection – Which Routes Are Safest?

- **Most accident-prone countries** (by total reported incidents):
  - **United States** – Highest volume of accidents due to **high traffic and extensive reporting**. The **FAA** has been quite the pace-setter
  - **Brazil**, **Mexico**, **Canada** – Have moderate accident frequencies.
  - **United Kingdom** – Incidents associated with **weather and airspace congestion**.
- **Regional aviation safety varies significantly** based on **infrastructure, regulations, and weather**.

Insight:

- **Prioritize safer, well-regulated routes** with **strong air traffic control** systems.
- Consider launching operations in **regions with moderate demand and low accident rates**.
- **Avoid high-risk zones** unless necessary, and invest in **risk mitigation strategies**.

4. Operational Risks – Which Flight Phases Are Most Dangerous?

- **Most accidents occur during**:
    1. **Landing** – Most critical phase (highest accident count).
    2. **Takeoff** – Second-most accident-prone phase.
    3. **Cruise & Maneuvering** – Fewer but often severe accidents.
- **Landing and takeoff accidents are often caused by**:
    - **Runway conditions, mechanical failures, or pilot errors**.
    - **Harsh weather, unstable approaches, or miscalculations**.

Insight:

- Invest in **pilot training for takeoff and landing safety procedures**.
- **Equip aircraft with advanced navigation systems**
- Ensure **runway safety checks** before landing and takeoff.

---

5. Weather & Environmental Impact – How Does Weather Affect Safety?

- **76,417 accidents** occurred under **VMC (good visibility)**.
- **IMC (low visibility) accidents were fewer but often more severe**.
- **Poor weather conditions lead to**:
    - **Navigation errors, turbulence, visibility issues, and emergency landings**.

Insight:

- Invest in **weather monitoring & predictive analytics**.
- Train pilots for operations in different conditions.
- **Plan alternate routes** for extreme weather conditions.

Final Recommendations

To successfully enter the aviation industry, the company should:

1. **Select the safest, most fuel-efficient aircraft** for long-term profitability.
2. **Begin with a small, strategically planned fleet** to test demand.
3. **Avoid high-risk routes & prioritize well-regulated airspaces**.
4. **Invest in advanced pilot training & navigation technology**.
5. **Implement real-time weather monitoring & route optimization systems**.
