Phase 1 Project-Aviation Safety Analysis by Wanjira_Nyambura

1.Business Understanding

The company is expanding into the aviation industry and exploring the purchase of aircraft for both commercial and private use. However, they lack clear insights into which aircraft types pose the least safety risks.

This project is designed to help the leadership team make data-driven decisions by analyzing historical aviation accident data from the National Transportation Safety Board (NTSB). The objective is to identify patterns in accident severity across different aircraft types, flight purposes, weather conditions, and phases of flight.

The final output includes safety recommendations and data-backed insights to guide aircraft procurement and inform risk mitigation strategies.

```
#Imports
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.impute import SimpleImputer
import seaborn as sns
```

First, I loaded the csv file and treated blanks and placeholders ie NaN and NULL as missing values. It is easy to amputate,clean or analyze them as missing values.

```
#load dataset and treat blanks,NaN,NULL as missing values
df=pd.read_csv("Aviation_Data.csv",na_values=['', ' ', 'NaN', 'NULL'])
#preview of the first 5
df.head()
```

```
/tmp/ipython-input-3-39574374.py:2: DtypeWarning: Columns (6,7,28) have mixed types. Specify dtype option on import or set low_memor
  df=pd.read_csv("Aviation_Data.csv",na_values=['', ' ', 'NaN', 'NULL'])
```

	Event.Id	Investigation.Type	Accident.Number	Event.Date	Location	Country	Latitude	Longitude	Airport.Code	Airpo
0	20001218X45444	Accident	SEA87LA080	1948-10-24	MOOSE CREEK, ID	United States	NaN	NaN	NaN	
1	20001218X45447	Accident	LAX94LA336	1962-07-19	BRIDGEPORT, CA	United States	NaN	NaN	NaN	
2	20061025X01555	Accident	NYC07LA005	1974-08-30	Saltville, VA	United States	36.922223	-81.878056	NaN	
3	20001218X45448	Accident	LAX96LA321	1977-06-19	EUREKA, CA	United States	NaN	NaN	NaN	
4	20041105X01764	Accident	CHI79FA064	1979-08-02	Canton, OH	United States	NaN	NaN	NaN	

5 rows × 31 columns

2. 🗒️ Data Understanding

The dataset was sourced from the National Transportation Safety Board (NTSB) and covers civil aviation accidents and incidents from 1962 to 2023 across the United States and international waters.

Dataset Overview:

- **Format:** CSV
- **Size:** 90,348 rows × 31 columns

Key Variables:

- **Aircraft.Make / Aircraft.Model:** Manufacturer and model of the aircraft
- **Event.Date:** Date of the incident

- **Injury.Severity:** Severity of injuries (Fatal, Serious, Minor, or None)
- **Aircraft.Damage:** Extent of damage (Destroyed, Substantial, Minor, None)
- **Purpose.of.Flight:** Type of operation (e.g. Personal, Instructional, Business)
- **Broad.Phase.of.Flight:** Phase of flight when the accident occurred (e.g. Landing, Takeoff)
- **Weather.Condition:** VMC (Visual Meteorological Conditions) or IMC (Instrument Meteorological Conditions)
- **Engine.Type:** Type of engine installed

Initial Observations:

- Several columns have significant missing or inconsistent values.
- I find features such as coordinates, IDs, and airport codes not relevant to the current business objective and will drop them.
- I will engineer additional columns, including:
  - `Year` (from `Event.Date`)
  - `Make_Model` (combining manufacturer and model)
  - `Severe_Injuries` (sum of fatal and serious injuries

3.Data Preparation

I started by checking the size/summary of our data frame. Tells how big the data is by calculating number of rows and columns.

```
df.shape
```

```
(90348, 31)
```

To understand data quality, I calculated the number and percentage of missing values in each column. This gave me a better insight on which columns I need to be drop or be impute.

```
missing_summary = df.isnull().sum().sort_values(ascending=False)
missing_percent = (df.isnull().mean() * 100).sort_values(ascending=False)
missing_df = pd.DataFrame({
    'Missing Count': missing_summary,
    'Missing Percent': missing_percent
})
missing_df
```

	Missing Count	Missing Percent
Schedule	77766	86.073848
Air.carrier	73700	81.573471
FAR.Description	58325	64.555939
Aircraft.Category	58061	64.263736
Longitude	55975	61.954886
Latitude	55966	61.944924
Airport.Code	40216	44.512330
Airport.Name	37644	41.665560
Broad.phase.of.flight	28624	31.681941
Publication.Date	16689	18.471909
Total.Serious.Injuries	13969	15.461327
Total.Minor.Injuries	13392	14.822686
Total.Fatal.Injuries	12860	14.233851
Engine.Type	8555	9.468942
Report.Status	7843	8.680878
Purpose.of.flight	7651	8.468367
Number.of.Engines	7543	8.348829
Total.Uninjured	7371	8.158454
Weather.Condition	5951	6.586753
Aircraft.damage	4653	5.150086
Registration.Number	2841	3.144508
Injury.Severity	2459	2.721698
Country	1685	1.865011
Amateur.Built	1561	1.727764
Model	1551	1.716695
Make	1522	1.684597
Location	1511	1.672422
Event.Date	1459	1.614867
Event.Id	1459	1.614867
Accident.Number	1459	1.614867
Investigation.Type	0	0.000000

Next steps: Generate code with missing_df View recommended plots New interactive sheet

Sometimes, missing or unclear values are keyed in as text e.g na,"n/a", "unknown", or "none". This step helped me identify the inconsistencies so I can treat them properly.

```
inconsistent_values = df.apply(lambda col: col[col.astype(str).str.strip().str.lower().isin(['unknown', 'n/a', 'na', 'none'])].count())
inconsistent_values
```

	0
Event.Id	0
Investigation.Type	0
Accident.Number	0
Event.Date	0
Location	2
Country	3
Latitude	0
Longitude	0
Airport.Code	1500
Airport.Name	223
Injury.Severity	0
Aircraft.damage	119
Aircraft.Category	14
Registration.Number	356
Make	27
Model	22
Amateur.Built	0
Number.of.Engines	0
Engine.Type	2053
FAR.Description	22
Schedule	0
Purpose.of.flight	6802
Air.carrier	18
Total.Fatal.Injuries	0
Total.Serious.Injuries	0
Total.Minor.Injuries	0
Total.Uninjured	0
Weather.Condition	0
Broad.phase.of.flight	548
Report.Status	0
Publication.Date	0

dtype: int64

Next, I combined both missing and inconsistent value checks to flag the top problematic columns for easier cleaning.

```
missing_df['Inconsistent Count'] = inconsistent_values
missing_df = missing_df[missing_df['Missing Count'] > 0]
missing_df.head(10)  # Display top 10 problematic columns
```

	Missing Count	Missing Percent	Inconsistent Count
Schedule	77766	86.073848	0
Air.carrier	73700	81.573471	18
FAR.Description	58325	64.555939	22
Aircraft.Category	58061	64.263736	14
Longitude	55975	61.954886	0
Latitude	55966	61.944924	0
Airport.Code	40216	44.512330	1500
Airport.Name	37644	41.665560	223
Broad.phase.of.flight	28624	31.681941	548
Publication.Date	16689	18.471909	0

Next steps: Generate code with missing_df View recommended plots New interactive sheet

The Event.Date was stored as a string which makes it hard to analyse trends over time. I converted it to datetime object using *pd.to_datetime() *, then just extracted the year to help group incidents by year without dealing with full dates complexity. I put the new extracts into new column and named it Year.

```
# Convert "Event.Date" from categorical to float defined as "Year"
df['Event.Date'] = pd.to_datetime(df['Event.Date'], errors='coerce')
df['Year'] = df['Event.Date'].dt.year
```

Here, I dropped columns that had alot of missing values and those I didn't find useful for our analysis goals

```
df.drop(columns=['Investigation.Type','Accident.Number','Latitude','Longitude','Airport.Code','Airport.Name','Air.carrier',
                 'FAR.Description','Publication.Date','Schedule','Registration.Number',
                 'Location','Amateur.Built', 'Report.Status', 'Event.Date'], inplace=True)
```

```
#confirm that the columns have dropped
df.shape
```

```
(90348, 17)
```

```
df.head()
```

	Event.Id	Country	Injury.Severity	Aircraft.damage	Aircraft.Category	Make	Model	Number.of.Engines	Engine.Type	Pu
0	20001218X45444	United States	Fatal(2)	Destroyed	NaN	Stinson	108-3	1.0	Reciprocating	
1	20001218X45447	United States	Fatal(4)	Destroyed	NaN	Piper	PA24-180	1.0	Reciprocating	
2	20061025X01555	United States	Fatal(3)	Destroyed	NaN	Cessna	172M	1.0	Reciprocating	
3	20001218X45448	United States	Fatal(2)	Destroyed	NaN	Rockwell	112	1.0	Reciprocating	
4	20041105X01764	United States	Fatal(1)	Destroyed	NaN	Cessna	501	NaN	NaN	

Next steps: Generate code with df View recommended plots New interactive sheet

Before diving into EDA, I needed to understand how much data I was working with and what was still incomplete. The command df.isnull().sum() returns a each column with its missing values.

```
df.isnull().sum()
```

	0
Event.Id	1459
Country	1685
Injury.Severity	2459
Aircraft.damage	4653
Aircraft.Category	58061
Make	1522
Model	1551
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
Year	1459

dtype: int64

I did Imputation with the mean to fill the missing values in the columns with numerical values. For the year, I used mode since it represents a discrete value and sing the mean would bring decimals and make the work messier.

I then converted all the imputated columns into integers for consistency and to remove the decimals points.

```
# Imputation with the mean was done to fill the missing values in the columns with numerical values
df['Total.Serious.Injuries'].fillna(df['Total.Serious.Injuries'].mean(), inplace=True)
df['Total.Fatal.Injuries'].fillna(df['Total.Fatal.Injuries'].mean(), inplace=True)
df['Total.Minor.Injuries'].fillna(df['Total.Minor.Injuries'].mean(), inplace=True)
df['Total.Uninjured'].fillna(df['Total.Uninjured'].mean(), inplace=True)
df['Number.of.Engines'].fillna(df['Number.of.Engines'].mean(), inplace=True)
df['Year'].fillna(df['Year'].mode()[0], inplace=True)
df['Year'] = df['Year'].astype(int)
df['Total.Serious.Injuries'] = df['Total.Serious.Injuries'].astype(int)
df['Total.Fatal.Injuries'] = df['Total.Fatal.Injuries'].astype(int)
df['Total.Minor.Injuries'] = df['Total.Minor.Injuries'].astype(int)
df['Total.Uninjured'] = df['Total.Uninjured'].astype(int)
df['Number.of.Engines'] = df['Number.of.Engines'].astype(int)
df.isnull().sum()
```

```
/tmp/ipython-input-13-963258328.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['Total.Serious.Injuries'].fillna(df['Total.Serious.Injuries'].mean(), inplace=True)
/tmp/ipython-input-13-963258328.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['Total.Fatal.Injuries'].fillna(df['Total.Fatal.Injuries'].mean(), inplace=True)
/tmp/ipython-input-13-963258328.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['Total.Minor.Injuries'].fillna(df['Total.Minor.Injuries'].mean(), inplace=True)
/tmp/ipython-input-13-963258328.py:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['Total.Uninjured'].fillna(df['Total.Uninjured'].mean(), inplace=True)
/tmp/ipython-input-13-963258328.py:6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['Number.of.Engines'].fillna(df['Number.of.Engines'].mean(), inplace=True)
/tmp/ipython-input-13-963258328.py:7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df['Year'].fillna(df['Year'].mode()[0], inplace=True)
```

	0
Event.Id	1459
Country	1685
Injury.Severity	2459
Aircraft.damage	4653
Aircraft.Category	58061
Make	1522
Model	1551
Number.of.Engines	0
Engine.Type	8555
Purpose.of.flight	7651
Total.Fatal.Injuries	0
Total.Serious.Injuries	0
Total.Minor.Injuries	0
Total.Uninjured	0
Weather.Condition	5951
Broad.phase.of.flight	28624
Year	0

dtype: int64

Next, I did Imputation with the mode to fill the missing values in the columns with categorical data.

```
df['Broad.phase.of.flight'].fillna(df['Broad.phase.of.flight'].mode()[0], inplace=True)
df['Country'].fillna(df['Country'].mode()[0], inplace=True)
df['Make'].fillna(df['Make'].mode()[0], inplace=True)
df['Aircraft.damage'].fillna(df['Aircraft.damage'].mode()[0], inplace=True)
df['Model'].fillna(df['Model'].mode()[0], inplace=True)
df['Engine.Type'].fillna(df['Engine.Type'].mode()[0], inplace=True)
df['Purpose.of.flight'].fillna(df['Purpose.of.flight'].mode()[0], inplace=True)
df['Weather.Condition'].fillna(df['Weather.Condition'].mode()[0], inplace=True)
df['Aircraft.Category'].fillna(df['Injury.Severity'].mode()[0], inplace=True)
df['Injury.Severity'].fillna(df['Injury.Severity'].mode()[0], inplace=True)
df.isnull().sum()
```

```
/tmp/ipython-input-14-4241630647.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['Broad.phase.of.flight'].fillna(df['Broad.phase.of.flight'].mode()[0], inplace=True)
/tmp/ipython-input-14-4241630647.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['Country'].fillna(df['Country'].mode()[0], inplace=True)
/tmp/ipython-input-14-4241630647.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['Make'].fillna(df['Make'].mode()[0], inplace=True)
/tmp/ipython-input-14-4241630647.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['Aircraft.damage'].fillna(df['Aircraft.damage'].mode()[0], inplace=True)
/tmp/ipython-input-14-4241630647.py:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['Model'].fillna(df['Model'].mode()[0], inplace=True)
/tmp/ipython-input-14-4241630647.py:6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['Engine.Type'].fillna(df['Engine.Type'].mode()[0], inplace=True)
/tmp/ipython-input-14-4241630647.py:7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['Purpose.of.flight'].fillna(df['Purpose.of.flight'].mode()[0], inplace=True)
/tmp/ipython-input-14-4241630647.py:8: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['Weather.Condition'].fillna(df['Weather.Condition'].mode()[0], inplace=True)
/tmp/ipython-input-14-4241630647.py:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['Aircraft.Category'].fillna(df['Injury.Severity'].mode()[0], inplace=True)
/tmp/ipython-input-14-4241630647.py:10: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chaine
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df['Injury.Severity'].fillna(df['Injury.Severity'].mode()[0], inplace=True)
```

	0
Event.Id	1459
Country	0
Injury.Severity	0
Aircraft.damage	0
Aircraft.Category	0
Make	0

Make	0
Model	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
Year	0

dtype: int64

```
#confirm the data is cleaned by checking first few rows.
df.head()
```

	Event.Id	Country	Injury.Severity	Aircraft.damage	Aircraft.Category	Make	Model	Number.of.Engines	Engine.Type	Pu
0	20001218X45444	United States	Fatal(2)	Destroyed	Non-Fatal	Stinson	108-3	1	Reciprocating	
1	20001218X45447	United States	Fatal(4)	Destroyed	Non-Fatal	Piper	PA24-180	1	Reciprocating	
2	20061025X01555	United States	Fatal(3)	Destroyed	Non-Fatal	Cessna	172M	1	Reciprocating	
3	20001218X45448	United States	Fatal(2)	Destroyed	Non-Fatal	Rockwell	112	1	Reciprocating	
4	20041105X01764	United States	Fatal(1)	Destroyed	Non-Fatal	Cessna	501	1	Reciprocating	

Next steps: Generate code with df | View recommended plots | New interactive sheet

Check for outliers

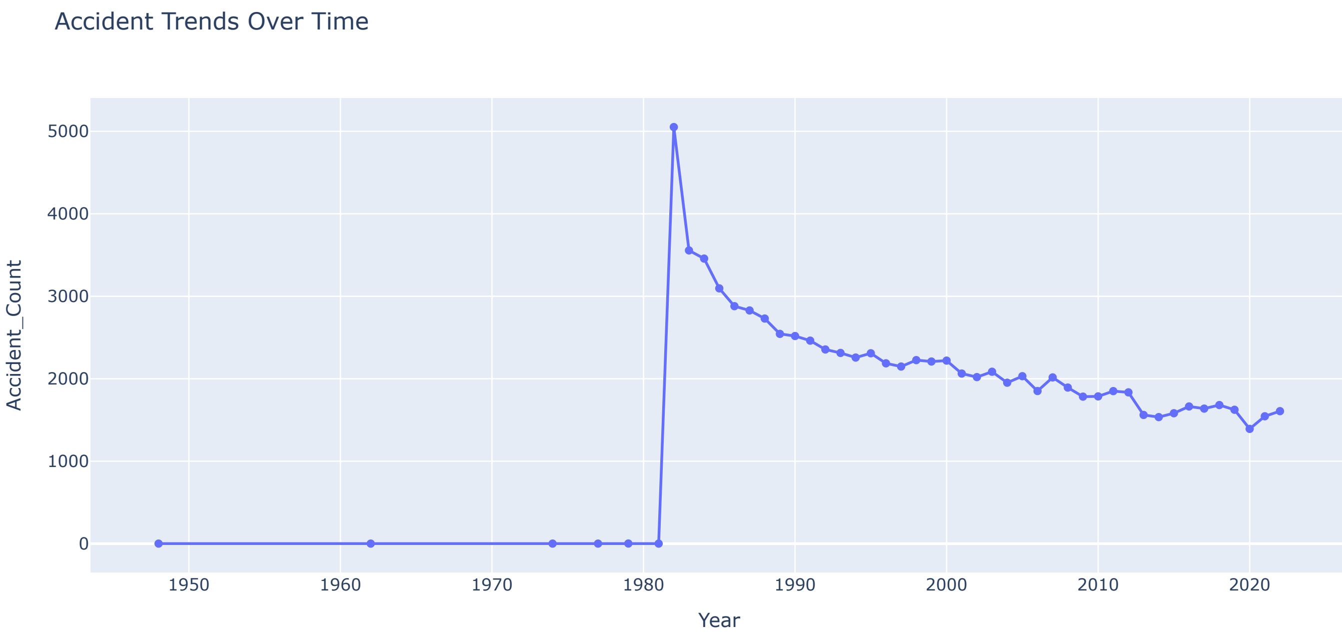
```
df.describe()
```

	Number.of.Engines	Total.Fatal.Injuries	Total.Serious.Injuries	Total.Minor.Injuries	Total.Uninjured	Year
count	90348.000000	90348.000000	90348.000000	90348.000000	90348.000000	90348.000000
mean	1.134347	0.555640	0.236607	0.304135	5.298889	1998.928798
std	0.429384	5.085584	1.423306	2.067189	26.750886	11.989644
min	0.000000	0.000000	0.000000	0.000000	0.000000	1948.000000
25%	1.000000	0.000000	0.000000	0.000000	0.000000	1988.000000
50%	1.000000	0.000000	0.000000	0.000000	1.000000	1998.000000
75%	1.000000	0.000000	0.000000	0.000000	2.000000	2009.000000
max	8.000000	349.000000	161.000000	380.000000	699.000000	2022.000000

4.**DATA ANALYSIS- with visuals**

I started by checking accident trends over the years. This helps me track when high and low accidents were recorded. We all know/ would assume that technological advancements over the years have led to a decline in number of accidents.

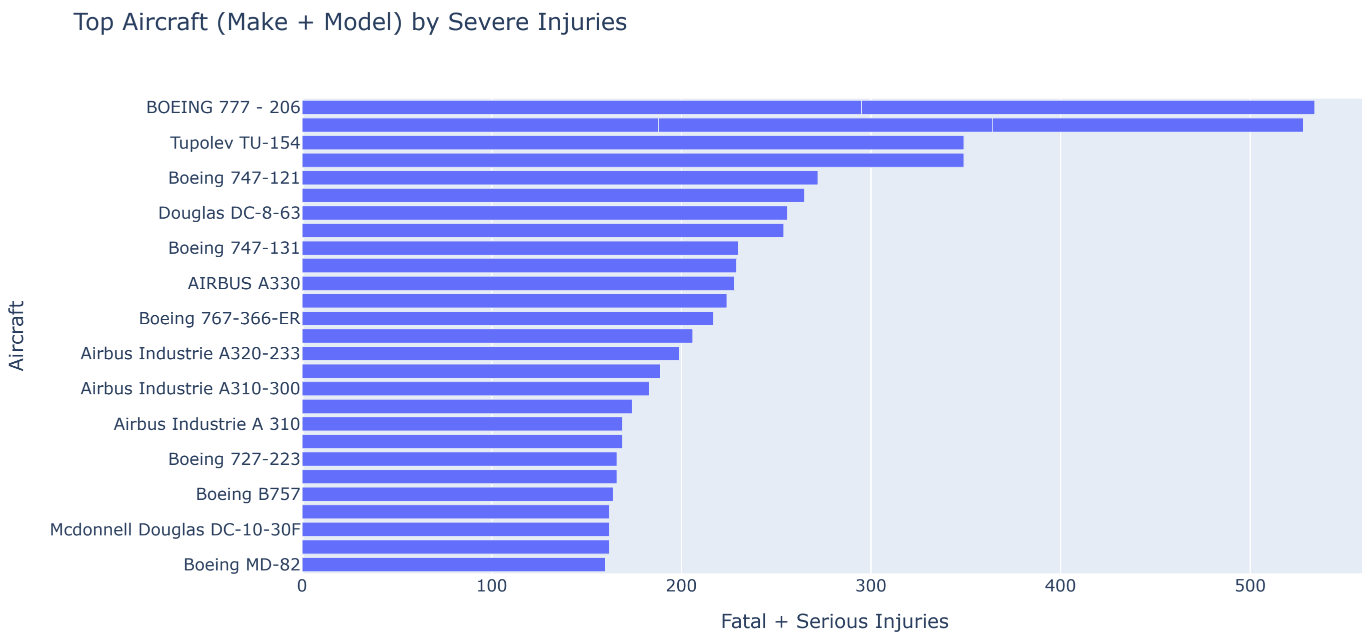
```
#Accident Trends Over Time
accidents_per_year = df.groupby('Year').size().reset_index(name='Accident_Count')

fig = px.line(accidents_per_year, x='Year', y='Accident_Count', markers=True, title='Accident Trends Over Time')
fig.show()
```



I created a new variable called (Severe Injuries), combining fatal and serious injuries, to rank aircraft risk.

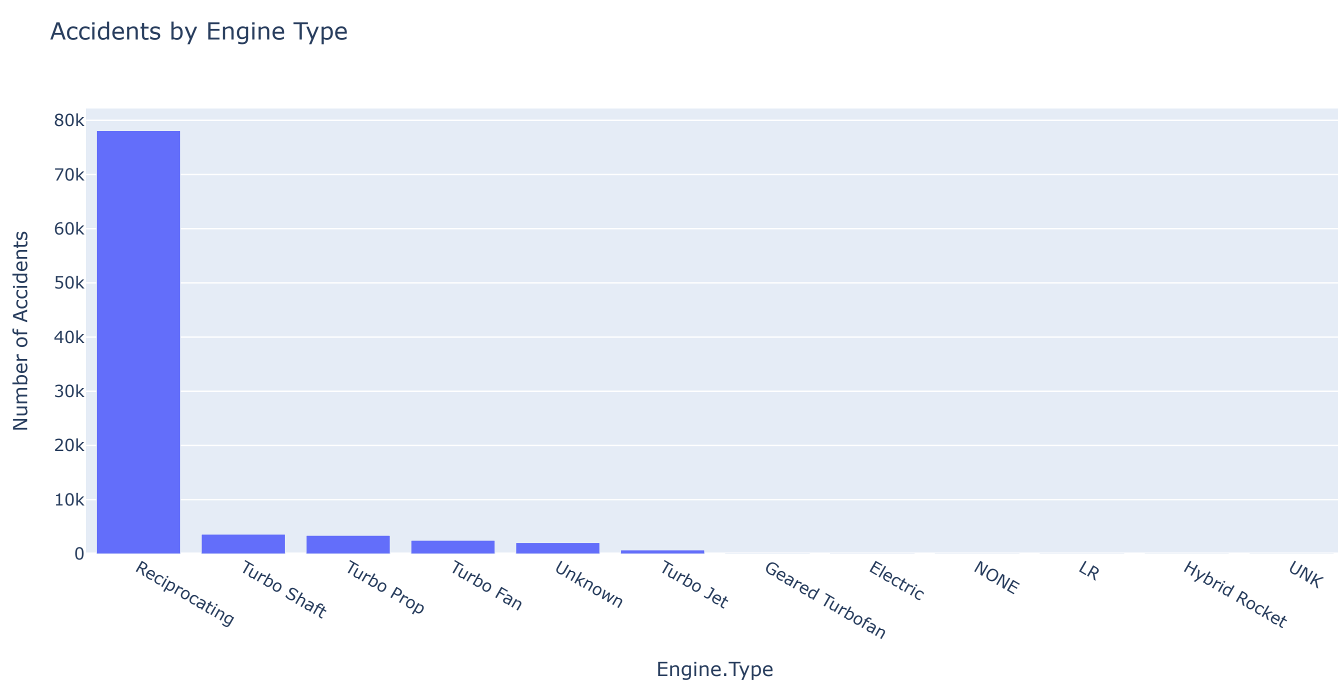
By grouping aircraft by make and model, I identified the top 30 aircraft types with the highest number of severe injury incidents. This will help us flag high-risk models to avoid purchasing.

```
#Aircraft Make & Model Risk Analysis
df['Severe_Injuries'] = df['Total.Fatal.Injuries'] + df['Total.Serious.Injuries']

df['Make_Model'] = df['Make'] + ' ' + df['Model']

# Sort by severe injuries and take top 15
top = df.sort_values(by='Severe_Injuries', ascending=False).head(30)

# Plot
fig = px.bar(top,
             x='Severe_Injuries',
             y='Make_Model',
             orientation='h',
             title='Top Aircraft (Make + Model) by Severe Injuries',
             labels={'Severe_Injuries': 'Fatal + Serious Injuries', 'Make_Model': 'Aircraft'})
fig.update_layout(yaxis=dict(categoryorder='total ascending'))
fig.show()
```

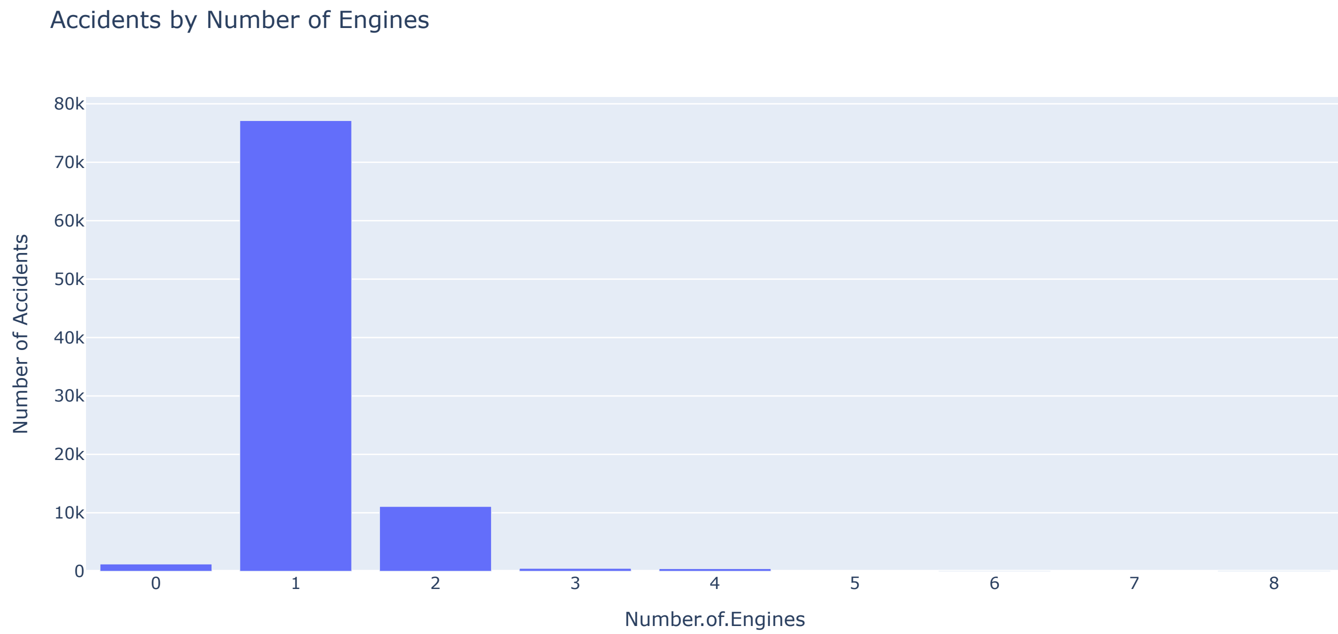


Engine type can influence both the complexity and risk level of any aircraft. Here, I looked at the number of accidents per engine type to determine if certain designs are more prone to accidents than the others.

```
#Accident Frequency by Engine Type
engine_type_counts = df['Engine.Type'].value_counts().reset_index()
engine_type_counts.columns = ['Engine.Type', 'Accident_Count']

fig = px.bar(engine_type_counts,
             x='Engine.Type',
             y='Accident_Count',
             title='Accidents by Engine Type',
             labels={'Accident_Count': 'Number of Accidents'})
fig.show()
```



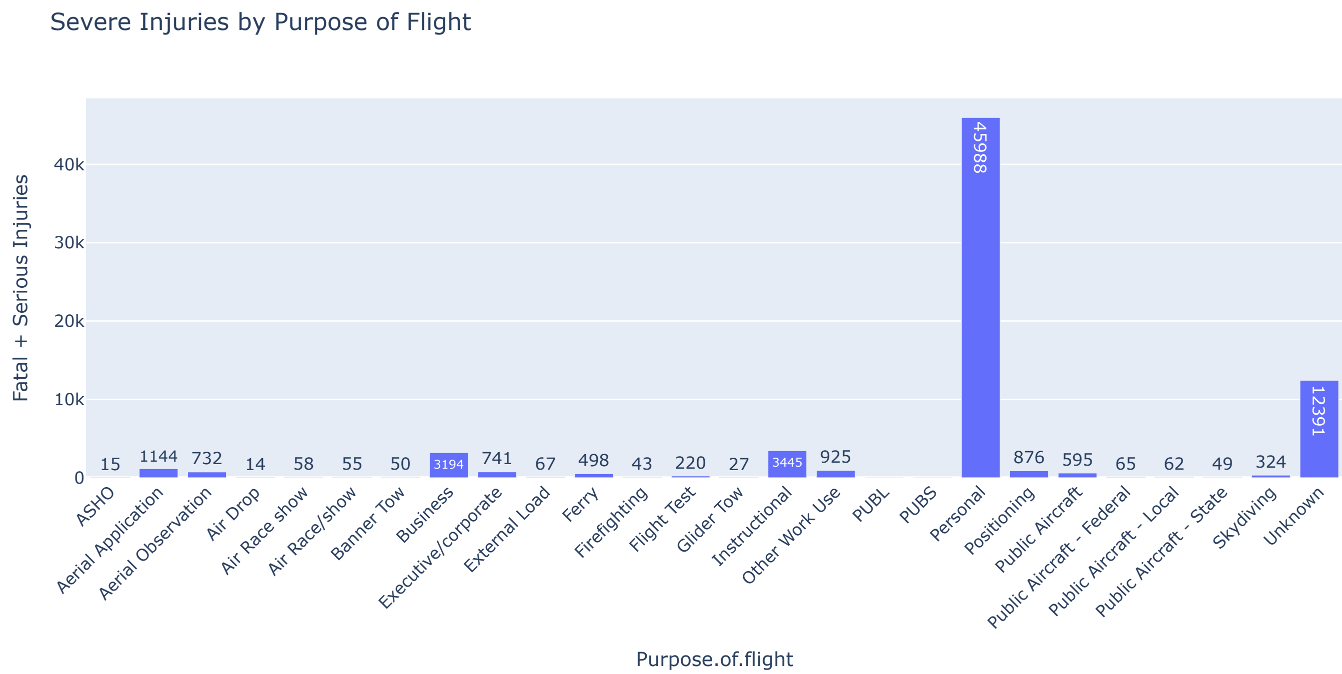
```
#Accident Frequency by Number of Engines
engine_num_counts = df['Number.of.Engines'].value_counts().reset_index()
engine_num_counts.columns = ['Number.of.Engines', 'Accident_Count']

fig = px.bar(engine_num_counts,
             x='Number.of.Engines',
             y='Accident_Count',
             title='Accidents by Number of Engines',
             labels={'Accident_Count': 'Number of Accidents'})
fig.show()
```



By grouping the data by* **(Purpose of Flight)*** I gained insight into which flight types are most commonly involved in serious or fatal incidents.

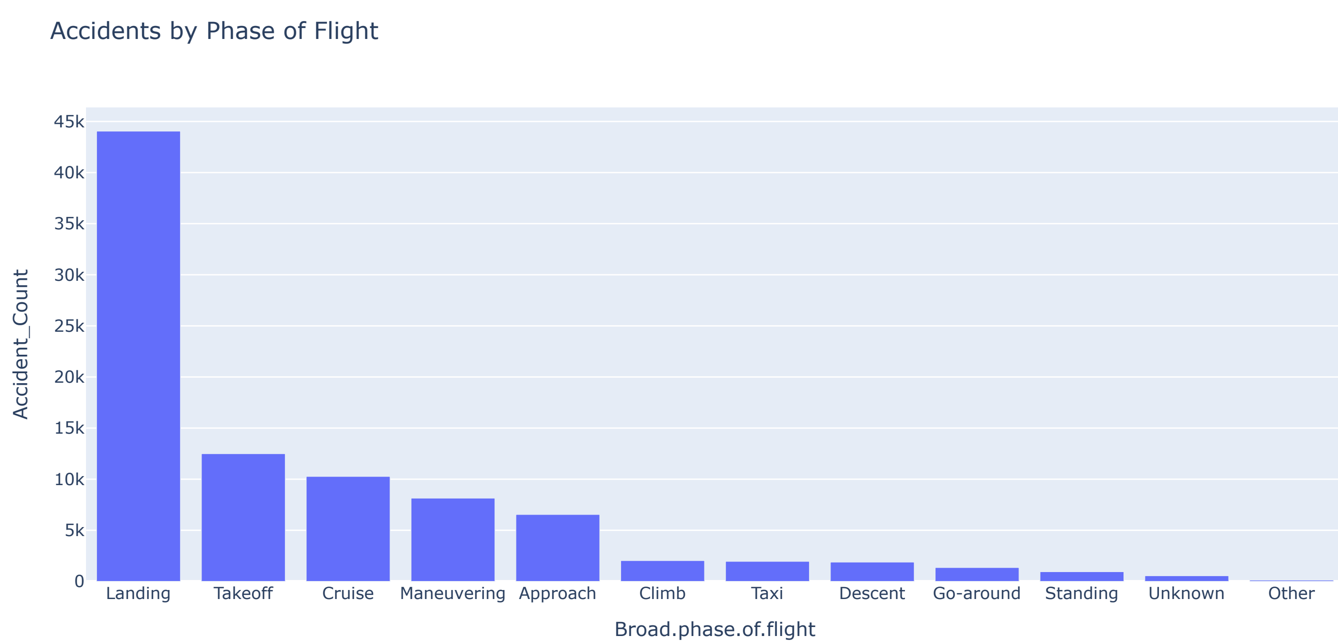
One would automatically assume that ***personal flights** * will have the highest number of severe injuries due to less safety protocols or less experienced pilots. From the data, instructional or business flights have relatively low severity, due to structured environments and regulatory protocals and oversight.

```
purpose_severity = df.groupby('Purpose.of.flight')['Severe_Injuries'].sum().reset_index()
fig = px.bar(purpose_severity,
             x='Purpose.of.flight',
             y='Severe_Injuries',
             title='Severe Injuries by Purpose of Flight',
             labels={'Severe_Injuries': 'Fatal + Serious Injuries'},
             text='Severe_Injuries')
fig.update_layout(xaxis_tickangle=-45)
fig.show()
```



Different phases of flight carry different risk levels. Here, I visualized the number of accidents occurring during each phase of the flight.

```
#Phase of Flight Risk

phase_counts = df['Broad.phase.of.flight'].value_counts().reset_index()
phase_counts.columns = ['Broad.phase.of.flight', 'Accident_Count']

fig = px.bar(phase_counts,
             x='Broad.phase.of.flight',
             y='Accident_Count',
             title='Accidents by Phase of Flight')
fig.show()
```



Next,I examined how **weather conditions** correlate with fatal injuries.

This gives us insight into how much the weather influences severity , with poor or unknown weather conditions showing higher risks, as expected.

```
# Weather Conditions and Risk

weather_risk = df.groupby('Weather.Condition')['Total.Fatal.Injuries'].mean().reset_index()

fig = px.bar(weather_risk,
             x='Weather.Condition',
             y='Total.Fatal.Injuries',
             title='Average Injuries by Weather Condition')
fig.show()
```