student

June 4, 2024

0.1 Phase 1 Project Submission

Please fill out: * Student name: Robert Sheynin * Student pace: self paced * Scheduled project review date/time: tbd * Instructor name: tbd * Blog post URL:tbd

1 Introduction & Business Problem

1.1 Objective

- The objective of this analysis is to determine which aircraft are the safest, lowest risk investment for a company expanding its line of business to include aviation. ## Intended Audience
- The intended audience for this analysis is a non-technical audience, such as a board of directors or investors or high-level executives who are considering investing in a new line of business.

2 Problem Statement

- Object of this analysis is to provide insight into which industries are the safest to invest in, based on the number of accidents and fatalities per year.

3 Data & Methodology

- The data used in this analysis is sourced from Kaggle.
- The data is suitable to perform an initial analysis of the safety of different aircraft types, and to determine which aircraft are the safest because it includes enough information to perform feature engineering to provide a suitable assessment of risk. ## Data Considerations and Limitations
- The data is limited to accidents from 1908-2009, and does not include more recent data.
- The data is limited to accidents, and does not include information on the number of successful flights for each aircraft type.
- The data includes missing values and in some instances irregularities in reporting (eg. Investigation.Type), and will require cleaning and imputation.

4 Setup

4.1 Import relevant libraries, load data, set viewing options

[1]:
```
## import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
import re

## read the aviation data set
df = pd.read_csv('data/Aviation_Data.csv')

# Set display options
pd.set_option('display.max_columns', None)  # Ensures all columns are shown
pd.set_option('display.max_colwidth', None)  # Ensures full content of each␣
 ↪cell is shown
pd.set_option('display.max_rows', None)  # Optionally display more rows
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_90625/3365175009.py:2
: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of
pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better
interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_90625/3365175009.py:1
1: DtypeWarning: Columns (6,7,28) have mixed types. Specify dtype option on
import or set low_memory=False.
  df = pd.read_csv('data/Aviation_Data.csv')
```

4.2 Data Cleaning

Here we will clean the data and make it ready for analysis. We will do the following: - Remove columns that are not needed for the analysis - Remove rows with missing values or impute them where it is possible or reasonable to do so - Remove duplicates - Convert columns to the correct data types - Check for and remove outliers - Standardize & Normalize numeric values and dates

4.2.1 Standardize Time Stamps and Dates

[2]:
```
df['Event.Date'] = pd.to_datetime(df['Event.Date'])
df['Year'] = df['Event.Date'].dt.year
df['Month'] = df['Event.Date'].dt.month_name()
df['Day'] = df['Event.Date'].dt.day_name()
```

4.2.2 Amateur Built

We will remove rows where the aircraft type is 'Amateur Built' because this is not a standard aircraft type and would be difficult to provide a meaningful analysis for.

[3]:
```
df = df[df['Amateur.Built'] == 'No']
```

4.2.3 Remove irrelevant columns, columns with too many missing values, columns with too many unique values, and columns with duplicate information to simply our dataset

[4]:
```
# drop rows dated before 1982 - too few samples
df = df[df['Year'] >= 1982]

# drop longitude and latitude columns - too many missing values
df = df.drop(['Latitude', 'Longitude'], axis=1)

# drop 'Amateur.Built' column - these are not considered in the analysis
df = df.drop(['Amateur.Built'], axis=1)

# drop Airport.Code column - too many unique values, not useful
df = df.drop(['Airport.Code'], axis=1)

# drop 'Airport.Name' column - too many unique values, not useful
df = df.drop(['Airport.Name'], axis=1)

# drop the 'Schedule' column - too many missing values
df = df.drop(['Schedule'], axis=1)

# drop the 'Air.Carrier' column - too many missing values
df = df.drop(['Air.carrier'], axis=1)

# drop the 'Report.Status' column - not useful
df = df.drop(['Report.Status'], axis=1)

# drop the Publication Date column - not useful
df = df.drop(['Publication.Date'], axis=1)

# drop the 'Registration.Number' column - too many unique values
df = df.drop(['Registration.Number'], axis=1)
```

```
# drop the 'Investigation.Type' column - in favor of 'Aircraft.Damage'
# df = df.drop(['Investigation.Type'], axis=1)

# drop the 'Far.Description' column - not useful
df = df.drop(['FAR.Description'], axis=1)
```

4.3 Aircraft Type

To focus our results we will only consider airplanes

4.3.1 Impute missing values for common aircraft makes

- Based on my research =Cessna=, =Piper= and =Beech= exclusively make airplanes.

[5]:
```
# update "Aircraft Category" to "Airplane" where "Make" is "Cessna", "Piper" or␣
 ↪"Beech"
df.loc[df['Make'].isin(['Cessna', 'Piper', 'Beech']), 'Aircraft.Category'] =␣
 ↪'Airplane'

# remove non-airplane records
df = df[df['Aircraft.Category'] == 'Airplane']
```

4.3.2 Purpose of Flight

Purpose of Flight is a categorical variable that describes the purpose of the flight. We will consider only the following categories: - Personal: This category includes flights conducted for personal reasons, not related to business activities or compensated work. These flights are typically for travel, leisure, or family reasons. - Instructional: Flights that are primarily for the purpose of flight training. These include both primary and advanced training sessions where an instructor is usually onboard teaching a student pilot. - Aerial Application: This involves flights conducted for agricultural purposes, such as crop dusting, seeding, or spraying pesticides. It can also include other types of aerial spraying such as mosquito control. - Business: Flights conducted for business purposes, where the travel is not just for commuting, but is integral to the business activity itself. This does not typically include airline operations but might involve the use of a private or corporate aircraft to attend meetings, site visits, etc. - Positioning: These are flights conducted to reposition the aircraft from one location to another without carrying passengers or cargo for profit. For example, moving an aircraft to a different airport for maintenance or to position for a subsequent commercial flight. - Ferry: Ferry flights involve moving aircraft from one location to another, typically after sales transactions, between leases, or to/from repair facilities. These flights usually do not carry commercial passengers or cargo. - Aerial Observation: This involves flights specifically for observing or monitoring from the air, such as surveillance, traffic monitoring, wildlife tracking, or patrol. - Public Aircraft: Flights operated by government or public entities for purposes like law enforcement, firefighting, military training, or other government-operated services.

And remove the following categories: - These categories are not actionable business recommendations - Unknown: This category is used when the purpose of the flight was not determined or not recorded at the time of the incident or accident report. - Other Work Use: This category encompasses flights conducted for commercial or work-related purposes other than those listed specifically, such as photography, pipeline inspection, or any other job that requires aerial mobility not covered

by more specific categories. - These categories have too few samples (less than or around 10) to be statistically significant: - PUBS - PUBL - ASHO

[6]:
```
# combine PUBS and Public Aircraft - State and PUBL and Public Aircraft -␣
 ↪Local, respectively - as these are likely mislabelled abbreviations
df['Purpose.of.flight'] = df['Purpose.of.flight'].replace({
    'PUBS': 'Public Aircraft - State',
    'PUBL': 'Public Aircraft - Local',
})

# drop rows where either purpose of flight is unknown, sample size is small␣
 ↪(less than or around 30), or are difficult to reconcile against the business␣
 ↪objective of finding low risk private sector investment opportunities.
df = df[~df['Purpose.of.flight'].isin(['Unknown', 'Other Work Use', 'ASHO',␣
 ↪'Air Drop', 'External Load', 'Air Race/show', 'Public Aircraft - Federal',␣
 ↪'Public Aircraft - Local', 'Public Aircraft - State', 'Public Aircraft'])]

# drop rows where purpose of flight is null
# df = df.dropna(subset=['Purpose.of.flight'])
```

[7]:
```
# if there are missing values in 'Total.Fatal.Injuries' but not in 'Injury.
 ↪Severity', then impute the missing values

# define a function to extract the number of fatalities or set to 0 for␣
 ↪'Non-Fatal'
def extract_fatalities(severity):
    if pd.isnull(severity):
        return None
    if 'Non-Fatal' in severity:
        return 0
    match = re.search(r'Fatal\((\d+)\)', severity)
    if match:
        return int(match.group(1))
    return None


# apply the function and impute missing values in 'Total.Fatal.Injuries'␣
 ↪directly
df['Total.Fatal.Injuries'] = df.apply(
    lambda row: extract_fatalities(row['Injury.Severity']) if pd.
 ↪isnull(row['Total.Fatal.Injuries']) else row['Total.Fatal.Injuries'],
    axis=1
)

# drop the 'Extracted.Fatalities', 'Injury.Severity' columns as they are no␣
 ↪longer needed
#df.drop(columns=['Injury.Severity'], inplace=True)
```

```
# replace NaN values in 'Total.Fatal.Injuries', 'Total.Serious.Injuries','Total.
 ↪Minor.Injuries' with 0
df['Total.Serious.Injuries'].fillna(0, inplace=True)
df['Total.Minor.Injuries'].fillna(0, inplace=True)
df['Total.Fatal.Injuries'].fillna(0, inplace=True)
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_90625/1487805048.py:2
5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['Total.Serious.Injuries'].fillna(0, inplace=True)
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_90625/1487805048.py:2
6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['Total.Minor.Injuries'].fillna(0, inplace=True)
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_90625/1487805048.py:2
7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['Total.Fatal.Injuries'].fillna(0, inplace=True)
```

4.3.3 Engine Type

Engine Type is a categorical variable that describes the type of engine used in the aircraft. We will consider only the following categories: - Reciprocating: This category includes aircraft with reciprocating engines, which are internal combustion engines that use one or more reciprocating pistons to convert pressure into a rotating motion. - Turbo Prop: This category includes aircraft with turbo propeller engines, which are gas turbine engines that drive a propeller to produce thrust. - Turbo Shaft: This category includes aircraft with turbo shaft engines, which are gas turbine engines that drive a shaft to produce thrust. - Turbo Jet: This category includes aircraft with turbojet engines, which are gas turbine engines that produce thrust by expelling exhaust gases at high speeds.

And remove the following categories: - These categories are not actionable business recommendations - None: This category is used when the engine type is not applicable or not recorded at the time of the incident or accident report. - Unknown: This category is used when the engine type was not determined or not recorded at the time of the incident or accident report. - These categories have too few samples (less than or around 10) to be statistically significant: - Geared Turbo Fan - Electric - Hybrid Rocket - Turbo Shaft

4.4 Feature Engineering

- Create a new column for non-fatal injuries
- Create a new column for the ratio of fatalities to non-fatal injuries

[8]:
```
# calculate non-fatal injuries
df['Total.Non.Fatal.Injuries'] = df['Total.Serious.Injuries'] + df['Total.Minor.
 ↪Injuries']

# calculate the ratio of fatal injuries to total injuries
df['Fatal_Injury_Ratio'] = df['Total.Fatal.Injuries'] / (df['Total.Fatal.
 ↪Injuries'] + df['Total.Non.Fatal.Injuries'])
```

[9]:
```
df = df[~df['Engine.Type'].isin(['Geared Turbofan', 'Unknown', 'Electric',␣
 ↪'Hybrid Rocket', 'Turbo Shaft', 'NONE', 'UNK', 'LR'])]
```

5 Export cleaned data

- Save cleaned data to a new csv file for analysis

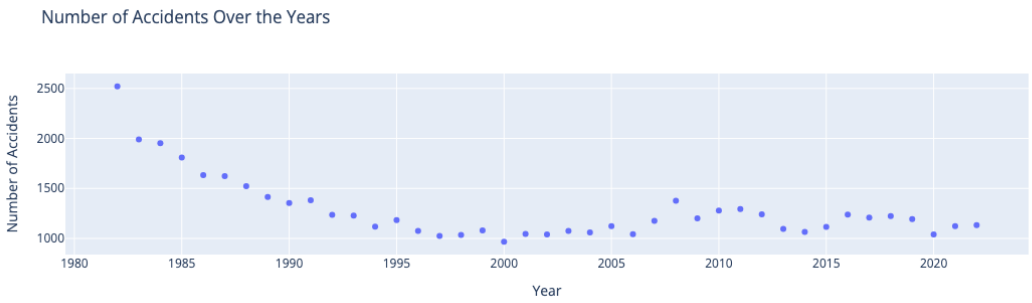
[10]:
```
df.to_csv('data/Aviation_Data_Cleaned.csv', index=False, mode='w')
```

6 Exploratory Data Analysis

6.1 Accidents over the years

[11]:
```
# Group by year and count the number of accidents
accidents_per_year = df.groupby('Year').size().reset_index(name='Accident␣
 ↪Count')
```

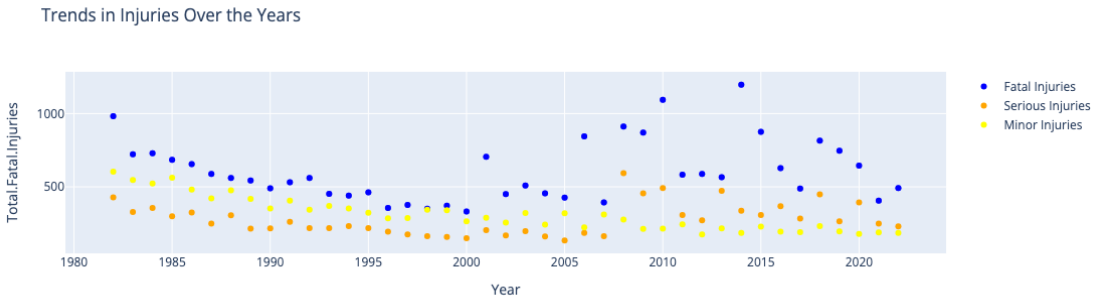
```
# Filter out any rows with NaN years if any
accidents_per_year = accidents_per_year.dropna()

# create a scatter plot of accidents over the years
fig = px.scatter(accidents_per_year, x='Year', y='Accident Count',
                 labels={'Year': 'Year', 'Accident Count': 'Number of␣
 ↪Accidents'},
                 title='Number of Accidents Over the Years')

fig.show()
```



6.2 Injuries over the Years

[12]:
```
# Group by year and sum the injuries for each type
injuries_per_year = df.groupby('Year')[['Total.Fatal.Injuries', 'Total.Serious.
 ↪Injuries', 'Total.Minor.Injuries']].sum().reset_index()

# Create a scatter plot for each type of injury
fig = px.scatter(injuries_per_year, x='Year', y='Total.Fatal.Injuries',
                 labels={'Year': 'Year', 'value': 'Number of Injuries'},
                 title='Trends in Injuries Over the Years')


fig.add_scatter(x=injuries_per_year['Year'], y=injuries_per_year['Total.Fatal.
 ↪Injuries'], mode='markers',
                 name='Fatal Injuries', marker=dict(color='blue'))
# Add serious injuries to the same figure
fig.add_scatter(x=injuries_per_year['Year'], y=injuries_per_year['Total.Serious.
 ↪Injuries'], mode='markers',
                 name='Serious Injuries', marker=dict(color='orange'))

# Add minor injuries to the same figure
```

```
fig.add_scatter(x=injuries_per_year['Year'], y=injuries_per_year['Total.Minor.
 ↪Injuries'], mode='markers',
                name='Minor Injuries', marker=dict(color='yellow'))


fig.show()
```



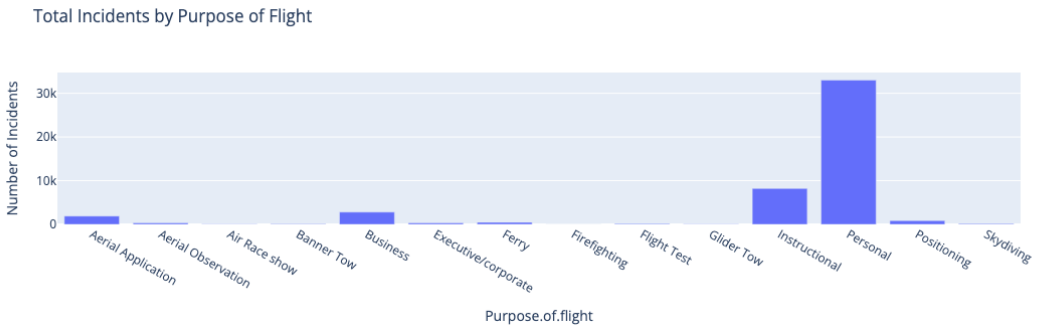
6.3 Findings

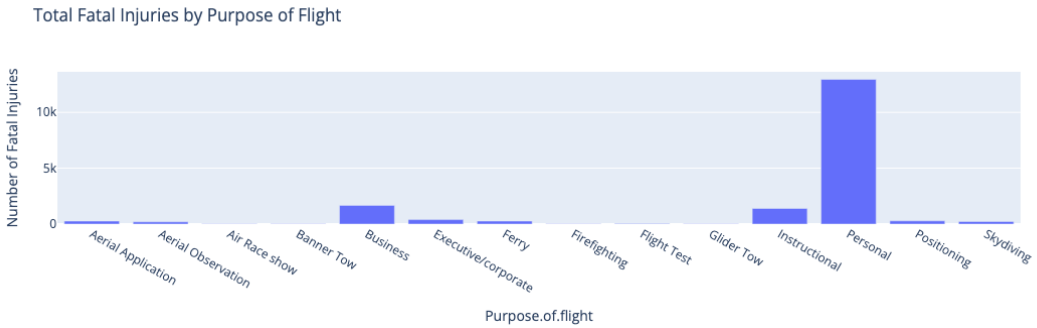
- The number of accidents and injuries has decreased over the years, which is a positive sign for the industry.
- The number of fatalities has also decreased over the years, which is a positive sign for the industry.

6.4 Compare total accidents & fatal injuries by purpose of flight

[13]:
```
# group the data by purpose of flight and calculate the total number of␣
 ↪incidents, fatal injuries, serious injuries, and minor injuries
safety_data = df.groupby('Purpose.of.flight').agg({
    'Total.Fatal.Injuries': 'sum',
    'Total.Serious.Injuries': 'sum',
    'Total.Minor.Injuries': 'sum',
    'Event.Id': 'count'
}).rename(columns={'Event.Id': 'Total.Incidents'}).reset_index()

# create a bar chart to visualize the total incidents by purpose of flight
fig_incidents = px.bar(safety_data, x='Purpose.of.flight', y='Total.Incidents',
                       title='Total Incidents by Purpose of Flight',
                       labels={'Total.Incidents': 'Number of Incidents'})
fig_incidents.show()

# create a bar chart to visualize the total fatal injuries by purpose of flight
fig_fatal = px.bar(safety_data, x='Purpose.of.flight', y='Total.Fatal.Injuries',
                   title='Total Fatal Injuries by Purpose of Flight',
```

```
                    labels={'Total.Fatal.Injuries': 'Number of Fatal Injuries'})
fig_fatal.show()
```

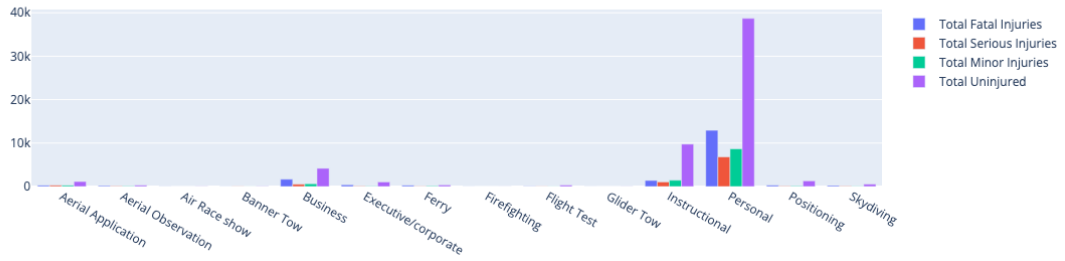




6.4.1 Compare types of injuries by purpose of flight

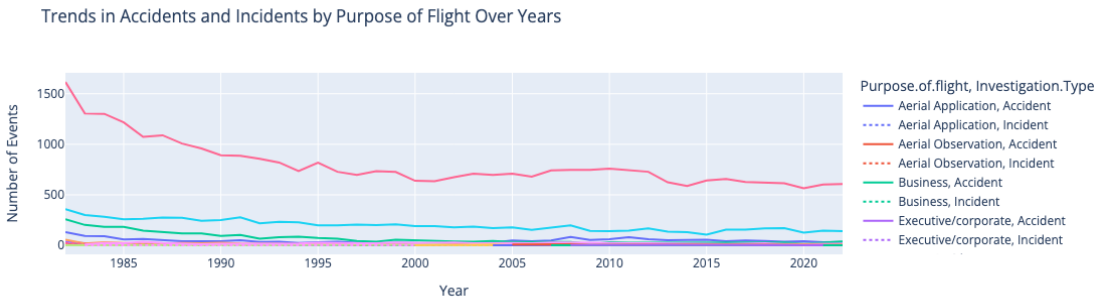
[14]:
```
# Create a stacked bar chart comparing the total number of fatal, serious,␣
 ↪minor injuries, and uninjured passengers by purpose of flight
injury_data = df.groupby('Purpose.of.flight').agg({
    'Total.Fatal.Injuries': 'sum',
    'Total.Serious.Injuries': 'sum',
    'Total.Minor.Injuries': 'sum',
    'Total.Uninjured': 'sum'
}).reset_index()

fig = go.Figure(data=[
    go.Bar(name='Total Fatal Injuries', x=injury_data['Purpose.of.flight'],␣
 ↪y=injury_data['Total.Fatal.Injuries']),
    go.Bar(name='Total Serious Injuries', x=injury_data['Purpose.of.flight'],␣
 ↪y=injury_data['Total.Serious.Injuries']),
```

```
    go.Bar(name='Total Minor Injuries', x=injury_data['Purpose.of.flight'],␣
 ↪y=injury_data['Total.Minor.Injuries']),
    go.Bar(name='Total Uninjured', x=injury_data['Purpose.of.flight'],␣
 ↪y=injury_data['Total.Uninjured'])
])

fig.show()
```

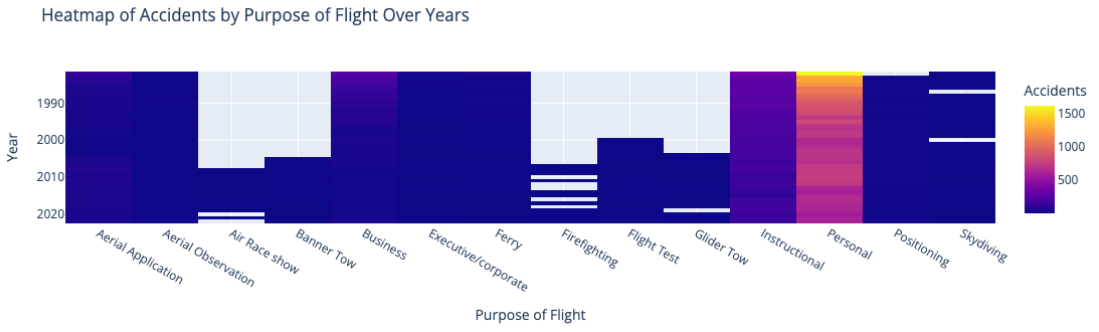


6.4.2 Chart the trend of accidents by purpose of flight over time

[15]:
```
# Group and aggregate data by year, purpose of flight, and investigation type
annual_safety_data = df.groupby(['Year', 'Purpose.of.flight', 'Investigation.
 ↪Type']).agg({
    'Total.Fatal.Injuries': 'sum',
    'Total.Serious.Injuries': 'sum',
    'Total.Minor.Injuries': 'sum'
}).reset_index()

# Count the number of incidents and accidents separately
annual_counts = df.groupby(['Year', 'Purpose.of.flight', 'Investigation.Type']).
 ↪size().reset_index(name='Counts')

fig = px.line(annual_counts, x='Year', y='Counts', color='Purpose.of.flight',
              line_group='Investigation.Type', line_dash='Investigation.Type',
              title='Trends in Accidents and Incidents by Purpose of Flight␣
 ↪Over Years',
              labels={'Counts': 'Number of Events'})
fig.show()
```

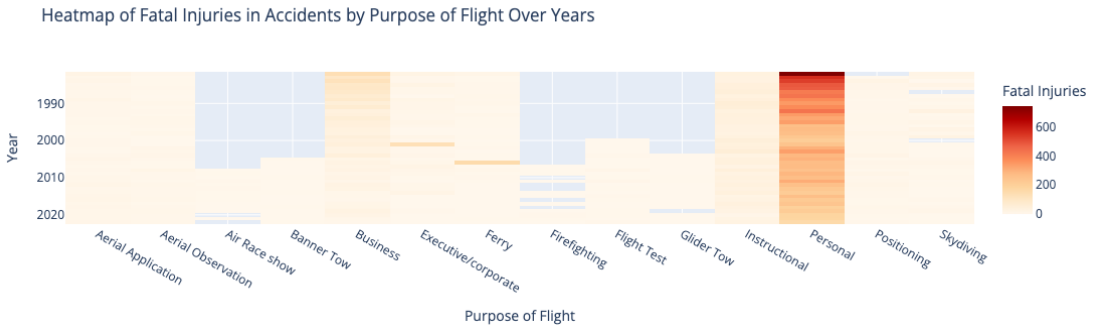


[16]:
```
# Filter data for accidents only
accidents_data = annual_counts[annual_counts['Investigation.Type'] ==␣
 ↪'Accident']

# Pivot table for heatmap
heatmap_data = accidents_data.pivot_table(
    values='Counts',
    index='Year',
    columns='Purpose.of.flight',
    aggfunc='sum'
)

# Create heatmap for accidents
fig = px.imshow(heatmap_data, aspect='auto', labels=dict(x="Purpose of Flight",␣
 ↪y="Year", color="Accidents"),
                title="Heatmap of Accidents by Purpose of Flight Over Years")
fig.show()
```



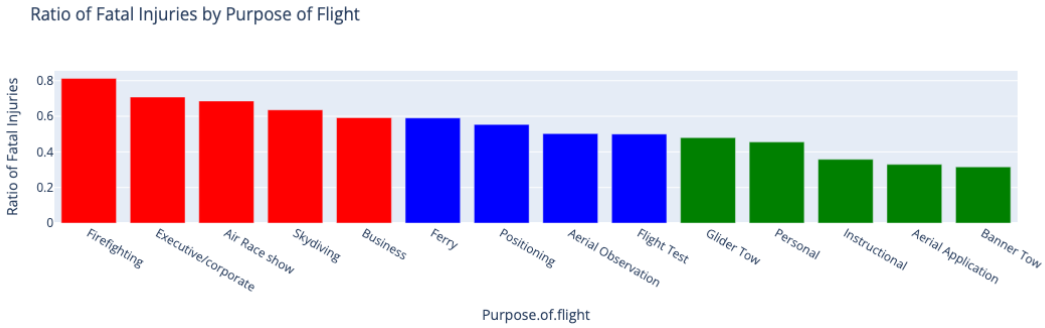
[17]:
```
# Filter for accidents and focus on fatal injuries
accidents_data = df[df['Investigation.Type'] == 'Accident']
```

```
# Group by Year and Purpose of Flight, then sum the fatal injuries
accidents_data = accidents_data.groupby(['Year', 'Purpose.of.flight']).agg({
    'Total.Fatal.Injuries': 'sum'
}).reset_index()

# Pivot table for heatmap using Total.Fatal.Injuries
heatmap_data = accidents_data.pivot_table(
    values='Total.Fatal.Injuries',  # Change from 'Counts' to 'Total.Fatal.
 ↪Injuries'
    index='Year',
    columns='Purpose.of.flight',
    aggfunc='sum'
)

# Create heatmap for fatal injuries in accidents
fig = px.imshow(heatmap_data, aspect='auto',
                labels=dict(x="Purpose of Flight", y="Year", color="Fatal␣
 ↪Injuries"),
                title="Heatmap of Fatal Injuries in Accidents by Purpose of␣
 ↪Flight Over Years",
                color_continuous_scale='OrRd')  # Optional: using a color scale␣
 ↪that better represents severity

fig.update_layout(
    xaxis=dict(side="bottom")  # Ensuring x-axis labels are at the bottom for␣
 ↪better readability
)
fig.show()
```



6.4.3 Ratio of fatal injuries to total accidents by purpose of flight & summary

[18]:
```
# redisplay the data as ratios
# Calculate the ratio of fatal injuries to total injuries
safety_data['Fatal_Injury_Ratio'] = safety_data['Total.Fatal.Injuries'] / (
    safety_data['Total.Fatal.Injuries'] + safety_data['Total.Serious.Injuries']␣
 ↪+ safety_data['Total.Minor.Injuries'])

# Sort the data by the injury ratio
safety_data = safety_data.sort_values(by='Fatal_Injury_Ratio', ascending=False)

# Determine the top three most dangerous and safest by using head and tail
top_dangerous = safety_data.head(5)['Purpose.of.flight'].tolist()
top_safest = safety_data.tail(5)['Purpose.of.flight'].tolist()

# Function to assign colors
def assign_color(x):
    if x in top_dangerous:
        return 'red'  # Dangerous highlighted in red
    elif x in top_safest:
        return 'green'  # Safest highlighted in green
    else:
        return 'blue'  # Default color for other categories

# Apply the color function
safety_data['Color'] = safety_data['Purpose.of.flight'].apply(assign_color)

# Create a bar chart to visualize the ratio of fatal injuries by purpose of␣
 ↪flight
fig_ratio = px.bar(safety_data, x='Purpose.of.flight', y='Fatal_Injury_Ratio',
                   title='Ratio of Fatal Injuries by Purpose of Flight',
                   labels={'Fatal_Injury_Ratio': 'Ratio of Fatal Injuries'},
                   color='Color',  # Use the assigned colors
                   color_discrete_map="identity")  # Use exact colors specified␣
 ↪in the DataFrame

fig_ratio.show()
```

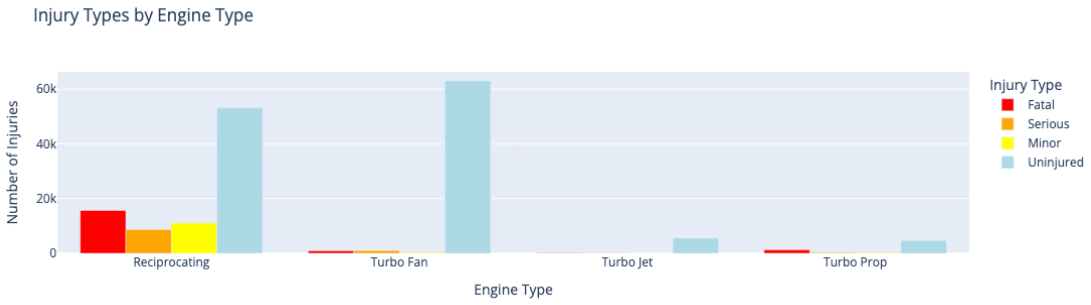


Findings I have attempted to normalize the data by using the ratio of fatal injuries to total injuries. - The riskiest enterprises are Fire Fighting, Executive/corporate, Air Race Show and Skydiving. - The safest enterprises are Instructional, Aerial Observation, and Banner Tow. ### Considerations - Fatal injuries are not the only measure of risk. Other factors such as the number of accidents, the number of injuries, and the number of fatalities should be considered. - Categories concerning Public Aircraft may be difficult to assess from a business investment perspective because they are not commercial enterprises.

6.5 Engine Type

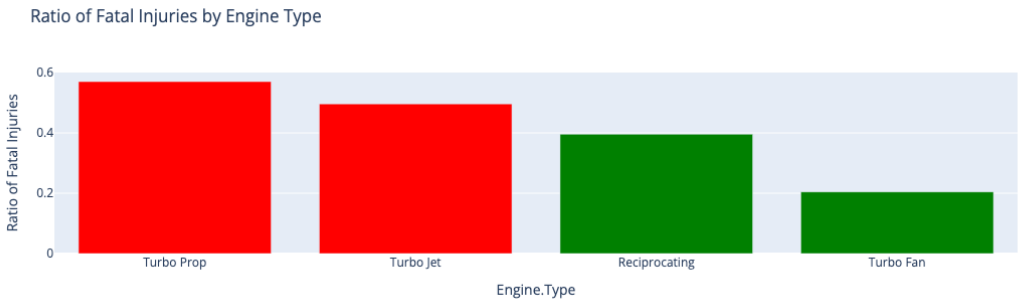
[19]:
```
# Preparing the data by grouping by 'Engine.Type' (or 'Number.of.Engines' if␣
 ↪using engine count)
grouped_data = df.groupby('Engine.Type')[['Total.Fatal.Injuries', 'Total.
 ↪Serious.Injuries',
                                          'Total.Minor.Injuries', 'Total.
 ↪Uninjured']].sum().reset_index()

# Creating a visualization where injuries are stacked and uninjured counts are␣
 ↪displayed separately
fig = go.Figure()

# Adding each injury type as a part of the stack
injury_types = ['Total.Fatal.Injuries', 'Total.Serious.Injuries', 'Total.Minor.
 ↪Injuries']
colors = ['red', 'orange', 'yellow']  # Colors for each bar segment

for injury, color in zip(injury_types, colors):
    fig.add_trace(go.Bar(
        name=injury.split('.')[1],  # Simplifying the name for the legend
        x=grouped_data['Engine.Type'],  # Using Engine.Type or Number.of.Engines
        y=grouped_data[injury],
        marker_color=color
    ))
```

```
# Adding a separate bar for uninjured next to the injury bars without offset
fig.add_trace(go.Bar(
    name='Uninjured',
    x=grouped_data['Engine.Type'],  # No offset, same x as injury bars for␣
 ↪grouping
    y=grouped_data['Total.Uninjured'],
    marker_color='lightblue'
))

# Adjusting layout to integrate stacked and grouped visualization effectively
fig.update_layout(
    barmode='group',  # Change to 'group' to place injury stack and uninjured␣
 ↪bar side by side
    title='Injury Types by Engine Type',
    xaxis_title='Engine Type',
    yaxis_title='Number of Injuries',
    legend_title_text='Injury Type'  # Adding a title to the legend for better␣
 ↪clarity
)

fig.show()
```



[20]:
```
#df['Fatal_Injury_Ratio'] = df['Total.Fatal.Injuries'] / (
#    df['Total.Fatal.Injuries'] + df['Total.Serious.Injuries'] + df['Total.
 ↪Minor.Injuries'])

# Group by Engine Type and calculate the average or total fatal injury ratio
safety_data = df.groupby('Engine.Type').agg({
    'Fatal_Injury_Ratio': 'mean'  # Use mean to get the average ratio per␣
 ↪engine type
}).reset_index()

# Sort the data by the injury ratio
```

```
safety_data = safety_data.sort_values(by='Fatal_Injury_Ratio', ascending=False)

# Determine the top three most dangerous and safest by using head and tail
top_dangerous = safety_data.head(2)['Engine.Type'].tolist()
top_safest = safety_data.tail(2)['Engine.Type'].tolist()

# Function to assign colors
def assign_color(x):
    if x in top_dangerous:
        return 'red'  # Dangerous highlighted in red
    elif x in top_safest:
        return 'green'  # Safest highlighted in green
    else:
        return 'blue'  # Default color for other categories

# Apply the color function
safety_data['Color'] = safety_data['Engine.Type'].apply(assign_color)

# Create a bar chart to visualize the ratio of fatal injuries by engine type
fig_ratio = px.bar(safety_data, x='Engine.Type', y='Fatal_Injury_Ratio',
                   title='Ratio of Fatal Injuries by Engine Type',
                   labels={'Fatal_Injury_Ratio': 'Ratio of Fatal Injuries'},
                   color='Color',  # Use the assigned colors
                   color_discrete_map="identity")  # Use exact colors specified␣
 ↪in the DataFrame

fig_ratio.show()
```



7 Findings

- The least dangerous engines are Turbo Fan and Reciprocating.
- The most dangerous engines are Turbo Jet and Turbo Prop. ## Considerations
- The number of accidents and injuries is relatively low for Turbo Fan engines, but the number

of fatalities is high. This could be due to the high number of flights in these categories but is becoming safer over the years.

[21]:
```
df['Purpose.of.flight'].value_counts()
```

[21]:
```
Purpose.of.flight
Personal                33016
Instructional            8214
Business                 2818
Aerial Application       1906
Positioning               857
Ferry                     487
Aerial Observation        363
Executive/corporate       351
Skydiving                 176
Flight Test               152
Banner Tow                 97
Air Race show              48
Glider Tow                 39
Firefighting               17
Name: count, dtype: int64
```