NTSB Accident Report Analysis

- Author: I Hull
- Date: October 2024
- Title: NTSB Accident Report Analysis USA 2000 to 2020

Source Files:

NTSB Air Accidents Data

State Population

State Name to Abbreviation Lookup

Background

This notebook uses data from the NTSB Air Accidents website for a 24 year period from 01/01/2001 to 01/01/2025.

The NTSB aviation accident database contains civil aviation accidents and selected incidents that occurred from 1962 to present within the United States, its territories and possessions, and in international waters. Foreign investigations in which the NTSB participated as an accredited representative will also be listed.

The data was downloaded as a JSON file and includes 39 fields detailing information relating to the accident or incident such as the date, the aircraft, the type of incident and whether there were any fatalities. After performing some data exploration, this project focuses on the actual report text rather than any other data relating to the incident.

Note: The data downloaded as a CSV seemed to contain different fields such as the make and model of aircraft but for the purposes of this report we will focus on the json file contents that contain the accident reports. Further analysis could be run to include this data and extract further interesting insights.

Research Aim and Motivation

Air incident reports contain a lot of information in summary textural form, including details relating to the conditions at the airport such as weather or runway, pilot actions, the particular conditions relating to the plane etc. This notebook investigates whether distinct topics or themes can be extracted from the text, to provide useful information on the main drivers of air accidents.

Method Notes

The data is cleaned to isolate fields of interest, drop rows containing null values and to extract information for accidents rather than incidents. Air accident: An event that results in serious injury, death, or destruction. Air incident: An event that compromises safety but doesn't result in serious injury, death, or destruction. Detailed method notes are contained within each section of the notebook, along with any interesting insights extracted from the data from the exploratory data analysis.

Topic Modelling

The data is analysed using Python libraries detailed in the import cell below. Traditional methods of extracting topics from text included Latent Dirchlet Allocation (LDA) and Non-Negative Matrix Factorisation (NMF). These are statistical methods that treat text as a bag of words and they can be complex to implement and interpret.

Newer methods include BERTopic, which uses BERT to create embeddings or representations of text. BERT is short for Bi-directional Encoder Representations, a type of large language model. These representations are clustered into easily interpretable topics. Some text cleaning is necessary to implement this model but less than older topic modelling libraries, since it is designed to work with large-language models which work with context.

1.0 Import libraries

In [125...

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

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

import warnings
warnings.filterwarnings("ignore")
```

In [126...

```
# Read in JSON file
df = pd.read_json(r'C:\Users\imoge\AllMLProjects\Data\AirAccidentData.json')
```

In [127...

```
# Shape of dataframe
df.shape
```

Out[127...

```
(42342, 39)
```

In [128...

```
# Top two rows
df.head(2)
```

Out[128...

	Oid	MKey	Closed	CompletionStatus	HasSafetyRec	HighestInjury	IsStudy
0	688c61a88126146d93a8e8a7	199500	False	In work	False	None	False
1	688c61a88126146d93a8e8a6	199498	True	Completed	False	None	False

2 rows × 39 columns

In [129…

```
# Columns
df.columns
```

Out[129…

```
Index(['Oid', 'MKey', 'Closed', 'CompletionStatus', 'HasSafetyRec',
       'HighestInjury', 'IsStudy', 'Mode', 'NtsbNumber',
       'OriginalPublishedDate', 'MostRecentReportType', 'ProbableCause',
       'City', 'Country', 'EventDate', 'State', 'Agency', 'BoardLaunch',
       'BoardMeetingDate', 'DocketDate', 'EventType', 'Launch', 'ReportDate',
       'ReportNum', 'ReportType', 'Vehicles', 'AirportId', 'AirportName',
       'AnalysisNarrative', 'FactualNarrative', 'PrelimNarrative',
       'FatalInjuryCount', 'MinorInjuryCount', 'SeriousInjuryCount',
       'InvestigationClass', 'AccidentSiteCondition', 'Latitude', 'Longitude',
       'DocketOriginalPublishDate'],
      dtype='object')
```

In [130…

```
df['EventType'].value_counts(normalize = True)
```

Out[130…

```
ACC    0.937863
INC    0.060602
OCC    0.001535
Name: EventType, dtype: float64
```

In [131…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42342 entries, 0 to 42341
Data columns (total 39 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Oid                    42342 non-null  object
 1   MKey                   42342 non-null  int64
 2   Closed                 42342 non-null  bool
 3   CompletionStatus       42342 non-null  object
 4   HasSafetyRec           42342 non-null  bool
 5   HighestInjury          41630 non-null  object
 6   IsStudy                42342 non-null  bool
 7   Mode                   42342 non-null  object
 8   NtsbNumber             42342 non-null  object
 9   OriginalPublishedDate  36687 non-null  object
 10  MostRecentReportType   40180 non-null  object
 11  ProbableCause          35712 non-null  object
 12  City                   42342 non-null  object
 13  Country                42314 non-null  object
 14  EventDate              42342 non-null  object
 15  State                  36706 non-null  object
 16  Agency                 41122 non-null  object
 17  BoardLaunch            42342 non-null  bool
 18  BoardMeetingDate       7 non-null      object
 19  DocketDate             20 non-null     object
 20  EventType              42342 non-null  object
 21  Launch                 20313 non-null  object
 22  ReportDate             33244 non-null  object
 23  ReportNum              93 non-null     object
 24  ReportType             42342 non-null  object
 25  Vehicles               42342 non-null  object
 26  AirportId              26155 non-null  object
```

```
 27  AirportName               26155 non-null  object
 28  AnalysisNarrative         38149 non-null  object
 29  FactualNarrative          35099 non-null  object
 30  PrelimNarrative           521 non-null    object
 31  FatalInjuryCount          42342 non-null  int64
 32  MinorInjuryCount          42342 non-null  int64
 33  SeriousInjuryCount        42342 non-null  int64
 34  InvestigationClass        0 non-null      float64
 35  AccidentSiteCondition     37297 non-null  object
 36  Latitude                  38284 non-null  float64
 37  Longitude                 38284 non-null  float64
 38  DocketOriginalPublishDate 23642 non-null  object
dtypes: bool(4), float64(3), int64(4), object(28)
memory usage: 11.5+ MB
```

Strange that several fields are shown as floats when they are strings

Data Cleaning

In [132...

```
# Check the date column
df['EventDate'].head(2)
```

Out[132...

```
0    2025-01-01T02:20:00Z
1    2024-12-31T14:30:00Z
Name: EventDate, dtype: object
```

In [133...

```
# Check the tail of the date
df['EventDate'].tail(2)
```

Out[133...

```
42340    2001-01-01T16:29:00Z
42341    2001-01-01T13:45:00Z
Name: EventDate, dtype: object
```

In [138...

```
# Set the event date to a datetime object
df['EventDate'] = pd.to_datetime(df['EventDate'])
```

In [139...

```
# Create a year and month column from the date
df['Year'] = df['EventDate'].dt.year
df['Month'] = df['EventDate'].dt.month

# Set to string
df['Year'] = df['Year'].astype('object')
df['Month'] = df['Month'].astype('object')
```

In [140...

```
# Check for nulls
df.isnull().sum()
```

Out[140...

```
Oid                          0
MKey                         0
Closed                       0
CompletionStatus             0
HasSafetyRec                 0
HighestInjury              712
IsStudy                      0
Mode                         0
NtsbNumber                   0
```

```
OriginalPublishedDate        5655
MostRecentReportType         2162
ProbableCause                6630
City                            0
Country                        28
EventDate                       0
State                        5636
Agency                       1220
BoardLaunch                     0
BoardMeetingDate            42335
DocketDate                  42322
EventType                       0
Launch                      22029
ReportDate                   9098
ReportNum                   42249
ReportType                      0
Vehicles                        0
AirportId                   16187
AirportName                 16187
AnalysisNarrative            4193
FactualNarrative             7243
PrelimNarrative             41821
FatalInjuryCount                0
MinorInjuryCount                0
SeriousInjuryCount              0
InvestigationClass          42342
AccidentSiteCondition        5045
Latitude                     4058
Longitude                    4058
DocketOriginalPublishDate   18700
Year                            0
Month                           0
dtype: int64
```

In [141...

```
# How many were recorded as accidents and how many as incidents?
df['EventType'].value_counts()
```

Out[141...

```
ACC    39711
INC     2566
OCC       65
Name: EventType, dtype: int64
```

In [142...

```
# Drop incidents and focus on accidents
df = df[df['EventType']=='ACC']
df.shape
```

Out[142...

```
(39711, 41)
```

In [143...

```
# Choose columns of interest
df = df[['ProbableCause','City','EventDate','Year','Month','State',
         'FactualNarrative', 'FatalInjuryCount', 'MinorInjuryCount',
         'SeriousInjuryCount','HighestInjury','AccidentSiteCondition',
         'Latitude', 'Longitude']]
```

In [144...

```
df.head()
```

Out[144…

	ProbableCause	City	EventDate	Year	Month	State	FactualNarrative	FatalInjuryCount	Mi
0	None	Naples	2025-01-01 02:20:00+00:00	2025	1	FL	None	0	
1	The student pilot's failure to maintain direct...	Anchorage	2024-12-31 14:30:00+00:00	2024	12	AK	None	0	
2	The pilot's failure to maintain airplane contr...	Los Alamos	2024-12-31 14:15:00+00:00	2024	12	NM	None	0	
3	The pilot's failure to maintain terrain cleara...	Galveston	2024-12-31 13:20:00+00:00	2024	12	TX	None	0	
4	None	Peebles	2024-12-30 16:15:00+00:00	2024	12	OH	None	1	

Data Exploration

In [145…

```
df.columns
```

Out[145…

```
Index(['ProbableCause', 'City', 'EventDate', 'Year', 'Month', 'State',
       'FactualNarrative', 'FatalInjuryCount', 'MinorInjuryCount',
       'SeriousInjuryCount', 'HighestInjury', 'AccidentSiteCondition',
       'Latitude', 'Longitude'],
      dtype='object')
```

In [146…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 39711 entries, 0 to 42341
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   ProbableCause          34858 non-null  object
 1   City                   39711 non-null  object
 2   EventDate              39711 non-null  datetime64[ns, UTC]
 3   Year                   39711 non-null  object
 4   Month                  39711 non-null  object
 5   State                  35646 non-null  object
 6   FactualNarrative       33432 non-null  object
 7   FatalInjuryCount       39711 non-null  int64
 8   MinorInjuryCount       39711 non-null  int64
 9   SeriousInjuryCount     39711 non-null  int64
 10  HighestInjury          39386 non-null  object
 11  AccidentSiteCondition  36389 non-null  object
 12  Latitude               37017 non-null  float64
 13  Longitude              37018 non-null  float64
dtypes: datetime64[ns, UTC](1), float64(2), int64(3), object(8)
memory usage: 4.5+ MB
```

We have a few null values for latitude and longitude but we won't worry about those at this point

Accidents by Year

In [149...

```
year_accidents = pd.DataFrame(df['Year'].value_counts().sort_index())
year_accidents = year_accidents.iloc[0:-1]
```

In [162...

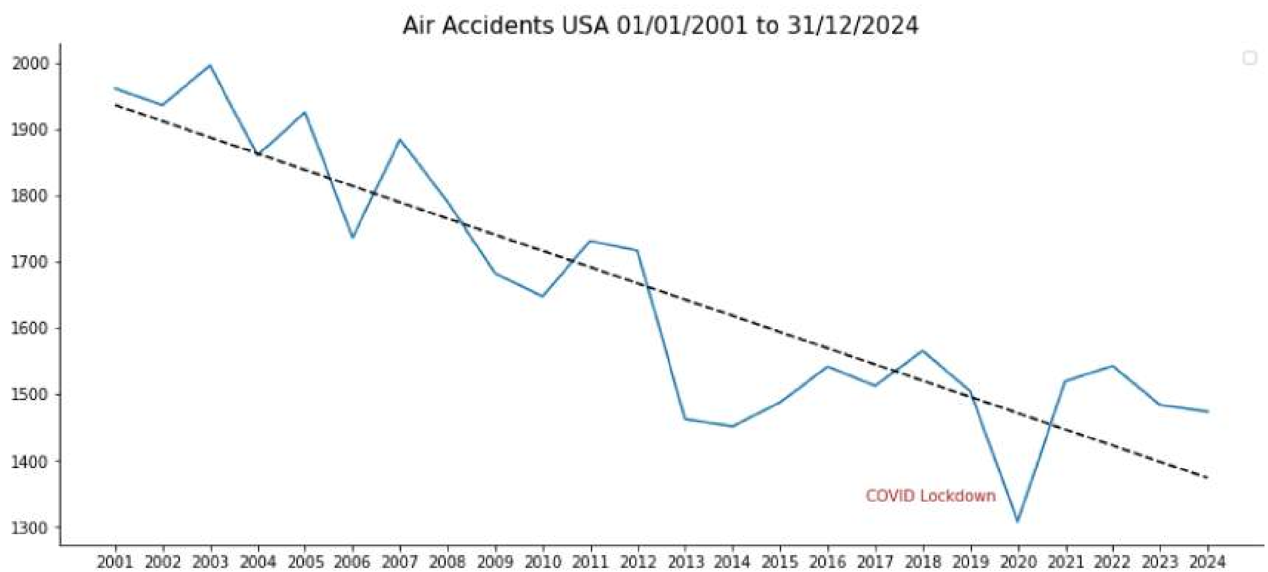
```
plt.figure(figsize = (14,6))
ax = sns.lineplot(data = year_accidents)

labels = year_accidents.index.astype('str')

#calculate equation for trendline
z = np.polyfit(year_accidents.index,year_accidents.Year.values, 1)
p = np.poly1d(z)

#add trendline to plot
plt.plot(year_accidents.index, p(year_accidents.index), color = 'k',linestyle='dashed')

# Title and legend
plt.title("Air Accidents USA 01/01/2001 to 31/12/2024", fontsize = 15)
ax.spines[['right', 'top']].set_visible(False)
ax.legend("")
ax.set_xticks(year_accidents.index)
plt.text(0.67,0.086,'COVID Lockdown', transform=ax.transAxes, color = 'firebrick')
ax.set_xticklabels(labels);
```



Trending down over the period and sharp fall in 2020 during Covid lockdown

Accidents by Month

In [267...

```
month_accidents = df.groupby('Month',as_index = False)['Year'].count()
month_accidents.columns = ['Month','Number']
month_accidents
```

Out[267…

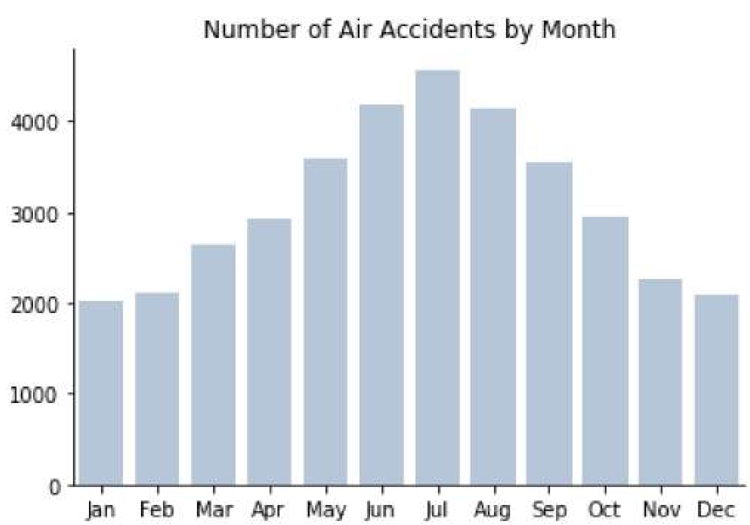
	Month	Number
0	1	2021
1	2	2104
2	3	2646
3	4	2928
4	5	3582
5	6	4192
6	7	4565
7	8	4131
8	9	3535
9	10	2959
10	11	2273
11	12	2080

In [268…

```
# Replace month numbers with month names (abbreviated or full)
import calendar
month_accidents['Month'] = month_accidents['Month'].apply(lambda x: calendar.month_abbr

month_order = list(calendar.month_abbr)[1:]  # ['Jan', 'Feb', ..., 'Dec']
month_accidents['Month'] = pd.Categorical(month_accidents['Month'], categories=month_or

ax = sns.barplot(data = month_accidents, x = 'Month', y = 'Number',color='lightsteelblu
ax.spines[['right', 'top']].set_visible(False)
plt.ylabel("")
plt.xlabel("")
plt.title("Number of Air Accidents by Month");
```



Accidents by Injury

In [163…

```
injuries = pd.DataFrame(df['HighestInjury'].value_counts(normalize = True)*100)
injuries
```

Out[163...

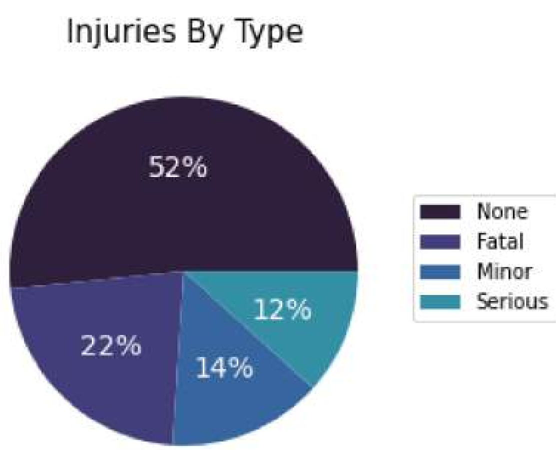
	HighestInjury
None	51.614787
Fatal	22.335348
Minor	14.355355
Serious	11.694511

In [164...

```
#define data
data = injuries.HighestInjury
labels = injuries.index

#define Seaborn color palette to use
colors = sns.color_palette('mako')[0:5]

#create pie chart
plt.pie(data,
        labels = labels,
        colors = colors,
        autopct='%.0f%%',
        textprops={'color':"w", 'fontsize': 14})
plt.title("Injuries By Type", fontsize = 15)
plt.legend(bbox_to_anchor=(1, 0.7))
plt.show()
```



In [165...

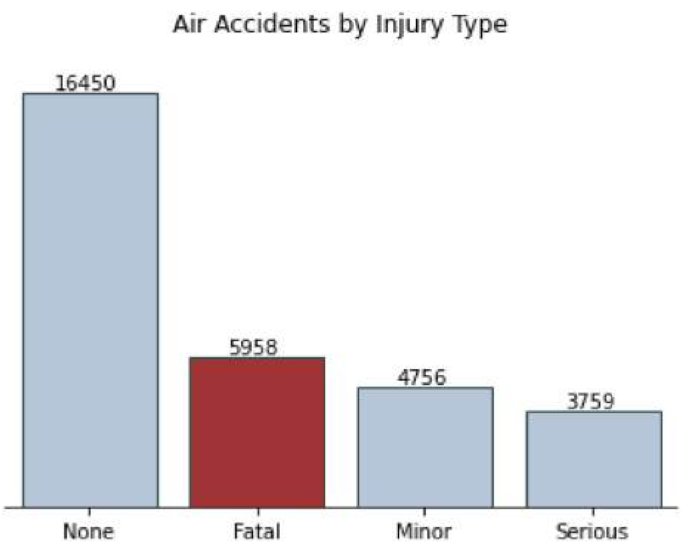
```
injuries = pd.DataFrame(df3['HighestInjury'].value_counts())
injuries
```

Out[165...

	HighestInjury
None	16450
Fatal	5958
Minor	4756
Serious	3759

In [167…

```
colors = ['lightsteelblue', 'firebrick', 'lightsteelblue', 'lightsteelblue']  # 'red' f
ax = sns.barplot(data = injuries, x = injuries.index,
                 y = 'HighestInjury',
                 palette = colors,
                 ec = 'darkslategray')
ax.spines[['right', 'top','left']].set_visible(False)
plt.title("Air Accidents by Injury Type",fontsize = 12, pad = 20)
plt.ylabel("Number",labelpad = 10)
plt.yticks([])
plt.ylabel("")
# Adding values on top of the bars
for p in ax.patches:
    ax.annotate(format(p.get_height(), '.0f'),
                     (p.get_x() + p.get_width() / 2.1, p.get_height()),
                     ha='center', va='bottom');
```



- Most accidents have no injuries
- When there are injuries the majority of fatal

In [168…

```
# Fatal injuries analysis
fatal = pd.DataFrame(df[df['FatalInjuryCount']>0]['FatalInjuryCount'].value_counts()).h
fatal.columns = ['Fatal Injury Count']
fatal['%'] = round(fatal['Fatal Injury Count']/fatal['Fatal Injury Count'].sum()*100,1)
fatal
```

Out[168…

	Fatal Injury Count	%
1	4453	51.4
2	2511	29.0
3	765	8.8
4	486	5.6
5	198	2.3
6	99	1.1
7	61	0.7

	Fatal Injury Count	%
9	34	0.4
8	32	0.4
10	25	0.3

In [175…

```
pd.DataFrame(df[df['FatalInjuryCount']>0]['FatalInjuryCount'].value_counts()).sort_inde
```

Out[175…

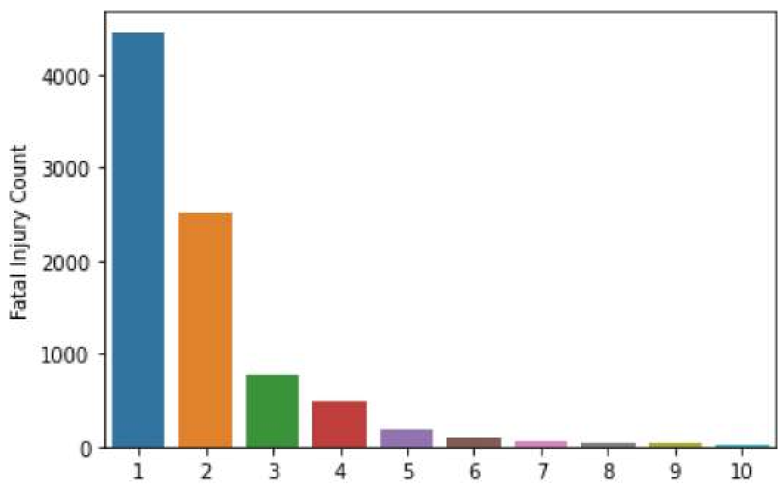
	FatalInjuryCount
298	1
265	1
228	1
224	1
206	1

In [170…

```
#create chart
sns.barplot(data = fatal, x = fatal.index, y = fatal['Fatal Injury Count'])
```

Out[170…

```
<AxesSubplot:ylabel='Fatal Injury Count'>
```



- Most fatal accidents involve 1 or 2 people and very few involve many
- In the 20 year period only one accident involved large numbers of people

In [171…

```
# Fatal accidents with more than 20 people involved
pd.DataFrame(df[df['FatalInjuryCount']>0]['FatalInjuryCount'].value_counts()).sort_inde
```

Out[171…

	FatalInjuryCount
298	1
265	1
228	1

	FatalInjuryCount
224	1
206	1
199	1
191	1
176	1
173	1
162	1

In [177…

```
# Find details on this flight
df[df['FatalInjuryCount']==265]
```

Out[177…

	ProbableCause	City	EventDate	Year	Month	State	FactualNarrative	FatalInjuryCount	N
40499	the in-flight separation of the vertical stabi...	Belle Harbor	2001-11-12 10:16:00+00:00	2001	11	NY	The Board's full report is available at http:/...	265	

American Airlines Flight 587 was a regularly scheduled international passenger flight from John F. Kennedy International Airport, New York City to Las Américas International Airport, Santo Domingo. On November 12, 2001, the Airbus A300B4-605R flying the route crashed into the neighborhood of Belle Harbor on the Rockaway Peninsula of Queens, New York City, shortly after takeoff, killing all 260 people aboard (251 passengers and 9 crew members), as well as 5 people on the ground.[1] It is the second-deadliest aviation accident in U.S. history, behind the crash of American Airlines Flight 191 in 1979,[a][1] and the second-deadliest aviation incident involving an Airbus A300, after Iran Air Flight 655.[1][3]

Accidents by Site Condition

In [178…

```
conditions = pd.DataFrame(df['AccidentSiteCondition'].value_counts(normalize = True)*10
conditions
```

Out[178…

	AccidentSiteCondition
VMC	93.816813
IMC	5.531892
Unknown	0.651296

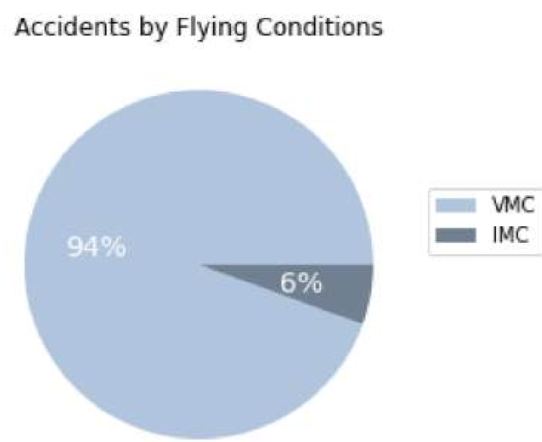
VMC = Visual Meterological Conditions
IMC = Instrument Meterological Conditions (Poor weather, Flying in clouds, Limited visibility)

In [179…

```
#define data
data = conditions.AccidentSiteCondition[0:2]
labels = conditions.index[0:2]

#define Seaborn color palette to use
colors = ['lightsteelblue','slategray']  # Replace with your desired colors

#create pie chart
plt.pie(data,
        labels = labels,
        colors = colors,
        autopct='%.0f%%',
        textprops={'color':"w", 'fontsize': 14})
plt.title("Accidents by Flying Conditions", fontsize = 12)
plt.legend(bbox_to_anchor=(1, 0.7))
plt.show()
```



Accidents by State

In [181…

```
# Accidents by state
state_accidents = pd.DataFrame(df['State'].value_counts()).reset_index()
state_accidents.columns = ['State','Accidents']
state_accidents['PerYear'] = round(state_accidents['Accidents']/24,0)
state_accidents.head()
```

Out[181…

	State	Accidents	PerYear
0	CA	3394	141.0
1	TX	2747	114.0
2	FL	2589	108.0
3	AK	2326	97.0
4	AZ	1412	59.0

In [182…

```
# Get a state name and abreviation lookup table
abrev = pd.read_excel(r'C:\Users\imoge\AllMLProjects\Data\Abbreviations.xlsx')
abrev.columns = ['Name','Abbrev']
```

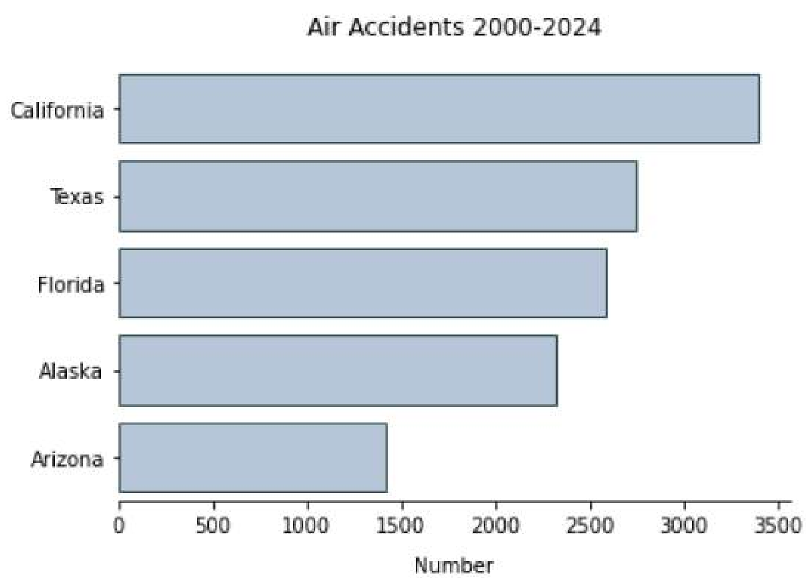
```
# Merge with the accident table to give names to the states
state_accidents = state_accidents.merge(abrev, how = 'left', left_on = 'State', right_o
state_accidents.drop(columns = 'Abbrev',axis = 1, inplace = True)
state_accidents.head()
```

Out[182...

	State	Accidents	PerYear	Name
0	CA	3394	141.0	California
1	TX	2747	114.0	Texas
2	FL	2589	108.0	Florida
3	AK	2326	97.0	Alaska
4	AZ	1412	59.0	Arizona

In [184...

```
# Plot the air accidents by state in total for 2000 to 2020
ax = sns.barplot(data = state_accidents[0:5], y = 'Name',
                 x = 'Accidents',
                 color = 'lightsteelblue',
                 ec = 'darkslategray')
ax.spines[['right', 'top','left']].set_visible(False)
ax.set_ylabel("")
ax.set_xlabel("Number", labelpad = 10)
ax.set_title("Air Accidents 2000-2024",fontsize = 12, pad = 15);
```



Most accidents are reported for California, followed by Texas, Florida. Most populous states so expect more flights

In [185...

```
# Get average state population from 2020 and 2023
pop = pd.read_excel(r'C:\Users\imoge\AllMLProjects\Data\USPopulation.xlsx')
pop['AveragePop'] = (pop['Pop. 2000'] + pop['Pop. 2023'])/2
pop = pop[['Name','AveragePop']]
pop.head()
```

Out[185...

	Name	AveragePop
0	United States	308169747.5
1	Alabama	4777837.5
2	Alaska	680169.5
3	Arizona	6280795.5
4	Arkansas	2870512.5

In [186...

```
# Merge the abreviation table with the state accidents table
combined = state_accidents.merge(pop, how = 'left', left_on = 'Name', right_on = 'Name'
combined.head()
```

Out[186...

	State	Accidents	PerYear	Name	AveragePop
0	CA	3394	141.0	California	36418423.0
1	TX	2747	114.0	Texas	25677164.5
2	FL	2589	108.0	Florida	19296648.5
3	AK	2326	97.0	Alaska	680169.5
4	AZ	1412	59.0	Arizona	6280795.5

In [187...

```
# Calculate the average annual accidents per million population
combined['AccperYearperMill'] = round(combined['PerYear']/combined['AveragePop']*100000
combined.sort_values(by = 'AccperYearperMill',ascending = False).head()
```

Out[187...

	State	Accidents	PerYear	Name	AveragePop	AccperYearperMill
3	AK	2326	97.0	Alaska	680169.5	142.6
38	WY	314	13.0	Wyoming	538921.5	24.1
9	ID	805	34.0	Idaho	1629341.5	20.9
29	MT	476	20.0	Montana	1017506.0	19.7
22	NM	587	24.0	New Mexico	1966694.0	12.2

In [188...

```
combined[combined['Name']=='California']
```

Out[188...

	State	Accidents	PerYear	Name	AveragePop	AccperYearperMill
0	CA	3394	141.0	California	36418423.0	3.9

In [189...

```
# Top five
top_five = combined.sort_values(by = 'AccperYearperMill',ascending = False).head(5)
top_five
```

Out[189...

	State	Accidents	PerYear	Name	AveragePop	AccperYearperMill
3	AK	2326	97.0	Alaska	680169.5	142.6
38	WY	314	13.0	Wyoming	538921.5	24.1
9	ID	805	34.0	Idaho	1629341.5	20.9
29	MT	476	20.0	Montana	1017506.0	19.7
22	NM	587	24.0	New Mexico	1966694.0	12.2

In [190...

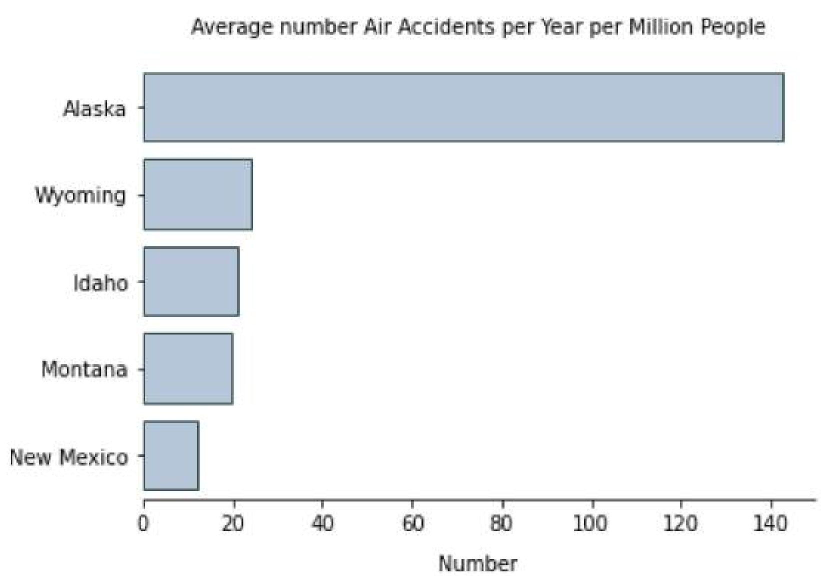
```
# Bottom five
bottom_five = combined.sort_values(by = 'AccperYearperMill',ascending = True).head()
bottom_five
```

Out[190...

	State	Accidents	PerYear	Name	AveragePop	AccperYearperMill
55	DC	4	0.0	District of Columbia	625529.0	0.0
48	PR	113	5.0	Puerto Rico	3507148.0	1.4
12	NY	758	32.0	New York	19274121.0	1.7
50	RI	57	2.0	Rhode Island	1072110.5	1.9
36	MA	320	13.0	Massachusetts	6675381.5	1.9

In [191...

```
ax = sns.barplot(data = top_five, y = 'Name',
                 x = 'AccperYearperMill',
                 color = 'lightsteelblue',
                 ec = 'darkslategray')
ax.spines[['right', 'top','left']].set_visible(False)
ax.set_ylabel("")
ax.set_xlabel("Number", labelpad = 10)
ax.set_title("Average number Air Accidents per Year per Million People",fontsize = 10,
```



In [98]:

```
# Split off data of interest
#acc_mill = top_ten[['Name','AccperYearperMill']]
```

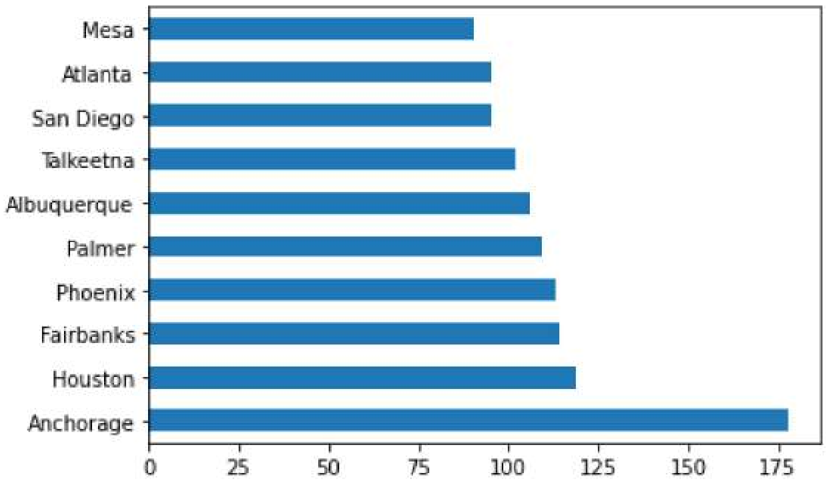
```
#acc_mill.columns = [['State','AccperYearMill']]
#acc_mill
```

Accidents by City

In [192...

```
df['City'].value_counts().head(10).plot(kind = 'barh')
```

Out[192... <AxesSubplot:>



Location of Accidents

In [193...

```
# Remove rows where we have no data on latitude and longitude
df = df[~df['Latitude'].isnull()]
df = df[~df['Longitude'].isnull()]
```

In [194...

```
from mpl_toolkits.basemap import Basemap
import folium
from folium import plugins
```

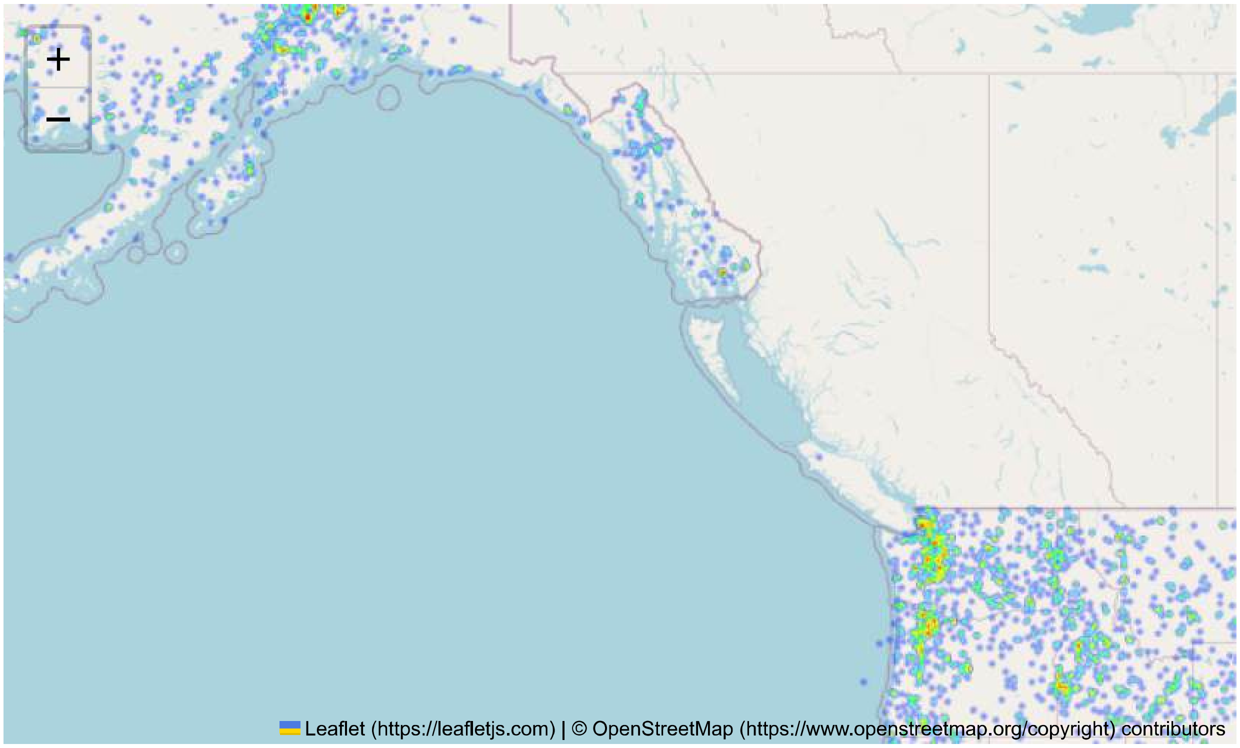
In [195...

```
# Convert latitude and longitude to lists
lat = df3['Latitude'].to_list()
long = df3['Longitude'].to_list()

# Create map centred on the US with a zoom
accident_map = folium.Map([38.9, -103], zoom_start=4.49)

heatmap = plugins.HeatMap(list(zip(lat,long)),radius = 2, blur = 1)
accident_map.add_child(heatmap)
```

Out[195...



Text Preprocessing

In [205...

```
# Find the reports that have completed
df2 = df[~df['FactualNarrative'].isnull()]
df2.shape
```

Out[205...

```
(32060, 14)
```

In [206...

```
# Look at summary of an example report
df2['ProbableCause'].iloc[3]
```

Out[206...

```
'The pilot’s improper fuel planning, which resulted in fuel exhaustion and a total loss
of power to both engines.'
```

In [209...

```
df2.iloc[3]['FactualNarrative']
```

Out[209...

```
'On November 27, 2024, about 0005 central standard time, a Cessna 340A airplane, N5757C,
sustained substantial damage when it was involved in an accident near Muskogee, Oklahom
a. The pilot and one passenger sustained minor injuries, and three passengers were not i
njured. The airplane was operated as a Title 14\xa0Code of Federal Regulations\xa0Part 9
1 personal flight.&#x0D;\nThe pilot reported that, before departing on the cross-country
flight, the airplane was “topped off” with 100 low lead fuel, resulting in a total of 18
3 gallons onboard for the flight. The pilot performed a preflight inspection of the airp
lane at night, but he did not have his headlamp with him, only his cellular phone. The p
ilot reported that, when verifying fuel levels in the dark, he typically feels inside th
e fuel tank with his finger, but he recalled not doing that with the right main fuel tan
k before departing on the accident flight.&#x0D;\nThe pilot’s fuel calculations showed t
hat the flight would require a total of 150 gallons of fuel. The airplane was equipped w
ith two main fuel tanks, each with a total capacity of 51 gallons, of which 50 gallons w
as usable; two auxiliary fuel tanks, each with a capacity of 32 gallons (31.5 gallons us
able each tank); and one left side locker fuel tank, which had a total capacity of 20.5
gallons (20 gallons usable).&#x0D;\nA review of ADS-B data showed that the airplane depa
```

rted from Mission Field Airport (LVM), Livingston, Montana, about 1840 on an instrument
flight rules flight plan to the intended destination of Muskogee-Davis Regional Airport
(MKO), Muskogee, Oklahoma.
\nThe pilot reported a higher than usual fuel consumptio
n rate during the first half of the flight, which was in instrument meteorological condi
tions (IMC). While operating in IMC, the pilot was operating the engines at 100° degrees
rich of peak versus the normal peak engine gas temperature. Additionally, the pilot was
operating the heater for the first half of the flight and then occasionally for the seco
nd half of the flight. The pilot reported that, while enroute, he transferred fuel from
the auxiliary fuel tanks and the left side locker fuel tank to the main fuel tanks. Afte
r the “final transfer” was completed, the pilot calculated that there was adequate fuel
remaining, including the required reserve fuel, to make the destination airport. Howeve
r, he reported that, over Tulsa (about 50 miles from the destination airport), the indic
ated fuel levels were less than he expected.
\nShortly after the pilot was cleared
for a visual approach at the destination, the right engine lost total power, followed by
the left engine. The pilot attempted to restart both engines to no avail. At the beginni
ng of the loss of engine power sequence, the pilot reported that the left side of the ai
rplane had about 30 lbs (about 4.84 gallons) of fuel remaining and the right side had ab
out 30 to 35 lbs (about 4.84 to 5.65 gallons) of fuel remaining.
\nDuring the subs
equent forced landing, the right wing impacted a permanent elevated static display on MK
O property, a U.S. Air Force T-33A jet trainer, located about 3,350 ft northwest of the
approach end of runway 13, and came to rest upright. The pilot and passengers egressed t
he airplane without further incident. The airplane sustained substantial damage to the f
uselage and the right wing.
\nDuring an onsite examination, fuel blighting was not
observed on the grass. The right main fuel tank and the right auxiliary fuel tank were s
eparated from the right wing during the impact sequence and sustained damage. All other
fuel tanks were found intact. All the fuel tank caps were found installed on their respe
ctive fuel tank. The fuel tank caps were removed to view the inside of the fuel tanks. F
uel was not observed in any of the fuel tanks. All the fuel tank bladders were found int
act, and they were not collapsed. The left side fuel selector was found on the left main
fuel tank. The right side fuel selector was found on the right auxiliary fuel tank. No s
igns of fuel leakage were observed on the airframe or on the two engines. During the rec
overy operation, no usable fuel was recovered from the airplane.
\nAccording to the
Cessna 340A Pilot’s Operating Handbook and Federal Aviation Administration (FAA) Approve
d Airplane Flight Manual, the airplane has an optional fuel low level warning light syst
em. This system provides a visual warning in the cockpit when the left main fuel tank or
the right main fuel tank, or both, contain about 60 lbs (about 9.68 gallons) of fuel. Th
e airplane was not equipped with this system, nor was it required to be.
\nThe con
figuration of the airplane’s fuel system did not allow the transfer of fuel from an auxi
liary fuel tank to either of the main fuel tanks; auxiliary fuel could only be provided
to the engine on the same side as the auxiliary fuel tank, via the fuel selector. The fu
el in a locker fuel tank was fed to the associated main fuel tank when the transfer pump
was activated by the pilot.
\nThe airplane was not equipped with a digital engine m
onitor or a digital fuel flow indicator system.
\nA review of the airplane mainten
ance records showed that the airplane’s most recent annual inspection was completed on O
ctober 3, 2023, at an airframe total time of 2,980.4 hours.
\nThe FAA Pilot’s Hand
book of Aeronautical Knowledge FAA-H-8083-25C discusses fuel tank quantity gauges and st
ates in part:
\nThe fuel quantity gauges indicate the amount of fuel measured by a
sensing unit in each fuel tank and is displayed in gallons or pounds. Aircraft certifica
tion rules require accuracy in fuel gauges only when they read “empty.” Any reading othe
r than “empty” should be verified. Do not depend solely on the accuracy of the fuel quan
tity gauges. Always visually check the fuel level in each tank during the preflight insp
ection, and then compare it with the corresponding fuel quantity indication.
\nThe
document also discusses fuel consumption rates and states in part:
\nThe rate of fu
el consumption depends on many factors: condition of the engine, propeller/rotor pitch,
propeller/ rotor revolutions per minute (rpm), richness of the mixture, and the percenta
ge of horsepower used for flight at cruising speed. The pilot should know the approximat
e consumption rate from cruise performance charts or from experience. In addition to the
amount of fuel required for the flight, there should be sufficient fuel for reserve. Whe
n estimating consumption, you must plan for cruise flight as well as startup and taxi, a
nd higher fuel burn during climb. Remember that ground speed during climb is less than d
uring cruise flight at the same airspeed. Additional fuel for adequate reserve should al
so be added as a safety measure. -'

In [210…

```
# Check for nulls
df2.ProbableCause.isnull().sum()
```

Out[210…

```
1057
```

In [211…

```
df2[df2['ProbableCause'].isnull()]
```

Out[211…

	ProbableCause	City	EventDate	Year	Month	State	FactualNarrative	FatalInjuryCount
7072	None	Lasne	2020-09-13 16:32:00+00:00	2020	9	None	The government of Belgium has notified the NTS...	2
7145	None	Teresina	2020-08-28 16:15:00+00:00	2020	8	None	The government of Brazil has notified the NTSB...	1
7158	None	Springs	2020-08-26 20:20:00+00:00	2020	8	None	The government of South Africa has notified th...	1
7250	None	Lobios (Ourense)	2020-08-08 10:20:00+00:00	2020	8	None	The Comision de Investigacion de Accidentes y ...	1
7291	None	Tarapaca-Amazonas	2020-07-30 18:03:00+00:00	2020	7	None	The government of Colombia has notified the NT...	1
...	...	...	...	...	...	...	...	...
28780	None	Fairmont	2007-10-26 19:00:00+00:00	2007	10	None	On October 26, 2007, about 1900, mountain dayl...	3
28799	None	Araya	2007-10-23 14:20:00+00:00	2007	10	None	On October 10, 2007, about 1420 Atlantic stand...	2
28816	None	Vancouver	2007-10-19 16:03:00+00:00	2007	10	None	On October 19, 2007, about 1603, pacific dayli...	1
28840	None	Bamfield	2007-10-13 15:00:00+00:00	2007	10	None	On October 13, 2007, about 1500 pacific daylig...	3
28842	None	Great Harbor	2007-10-13 10:36:00+00:00	2007	10	None	On October 13, 2007, at 1036 eastern daylight ...	1

1057 rows × 14 columns

In [223...

```
df2.iloc[3]['ProbableCause']
```

Out[223...

```
'The pilot’s improper fuel planning, which resulted in fuel exhaustion and a total loss
of power to both engines.'
```

In [224...

```
# Have a look at example of missing summary reports
df2['FactualNarrative'].loc[18111]
```

Out[224...

```
'During the landing flare to a private field, the airplane encountered a crosswind from
the southwest. The wind pushed the airplane to the north and the airplane landed prematu
rely on a high berm located to the north of the runway. The landing gear collapsed and t
he airplane was damaged by the postimpact fire. The pilot did not report any mechanical
anomalies with the airplane. -'
```

In [225...

```
# Drop the missing rows
df2 = df2[~df2['ProbableCause'].isnull()]
```

In [226...

```
# Get count of the words in the report and summary text
df2["count_narrative"] = df2['FactualNarrative'].str.split().apply(len)
df2["count_prob_cause"] = df2['ProbableCause'].str.split().apply(len)
```

In [227...

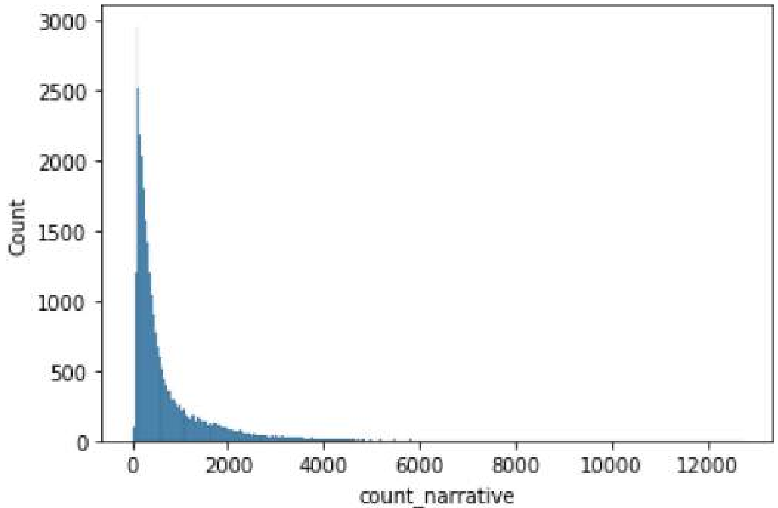
```
# Look at the distribution of the word count for the narrative
df2['count_narrative'].describe()
```

Out[227...

```
count    31003.000000
mean       685.914428
std        919.107312
min          3.000000
25%        175.000000
50%        344.000000
75%        784.000000
max      12730.000000
Name: count_narrative, dtype: float64
```

In [228...

```
# Plot distribution
sns.histplot(data = df2, x = 'count_narrative');
```



Right skewed with median length of 328 words. There are some long reports in the data of up to 12730 words which would be difficult for sentence transformer embeddings that work on text up to 256 tokens.

In [229…

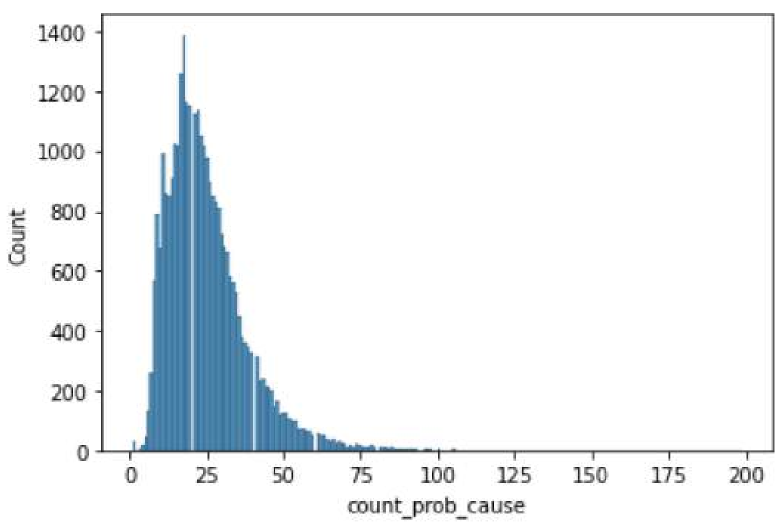
```
# Look at the distribution of the word count for the summary
df2['count_prob_cause'].describe()
```

Out[229…

```
count    31003.000000
mean        25.335548
std         13.891441
min          1.000000
25%         16.000000
50%         23.000000
75%         31.000000
max        199.000000
Name: count_prob_cause, dtype: float64
```

In [219…

```
# Plot distribution
sns.histplot(data = df2, x = 'count_prob_cause');
```



Some right skew with median of 23 words. The maximum length is 199 and a lot of the important reasons for the accident are summarised in this text

Observations

- The full reports do contain a lot of information which could be useful but some of them are very long and would need to be removed for BERTopic to work efficiently.
- These reports would probably work better with LDA which handles long texts.
- For this exercise the summary of the report in the column 'Probable Cause' is taken for training the model.
- This contains much shorter succinct text which would be more helpful in running BERTopic. It will also not require the level of text cleaning needed for the longer report which contains a lot of repetive information about the date and location, both of which are not required for this analysis as we have that information in the table data and also a lot of explanatory words which we might include as stop words and again are not information rich.
- The downside of using just BERTopic is that this throws out just one topic for a piece of text, rather than a range of probable topics which we get in LDA.
- Notebook 2 continues the work on this

In [230…

```
df2.to_csv(r'C:\Users\imoge\AllMLProjects\Data\USAAccident2001to2021Cleaned.csv')
```

In [221…

```
# Use a function to plot a wordcloud (we don't need stopwords)
def wordcloud(df,title):

    # Create text object
    text = " ".join(review for review in df3['ProbableCause'])

    # Set stopwords
    stopwords = set(STOPWORDS)
    stopwords = ["resulted"] + list(STOPWORDS)

    # Create and generate a word cloud image:
    wordcloud = WordCloud(max_words = 500,
                          colormap = 'Blues',
                          collocations = True,
                          background_color="Black",
                          stopwords = stopwords,
                          width=1000, height=600).generate(text)

    # Plot the wordcloud
    plt.figure(figsize = (10,8))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.title(title)
    plt.show()
```

In [222…

```
wordcloud(df2,'Accident Report Summary')
```

Accident Report Summary

In [ ]:

```
#import contractions
#from nltk.tokenize import word_tokenize
#import string
```

In [ ]:

```
# test code for text cleaning
#a = df["FactualNarrative"].apply(lambda x: x.strip().lower())
#b = a.apply(lambda x: contractions.fix(x))
#c = b.apply(lambda x: word_tokenize(x))
#punc = string.punctuation
#stop = ['january','february','march','april','may','june',
       # 'july','august','september','october','november','december']
#d = c.apply(lambda x: [w for w in x if w not in stop])
#e = d.apply(lambda x: [word for word in x if word not in punc])
#f = e.apply(lambda x: [n for n in x if not n.isnumeric()])
#g = f.apply(lambda x: [e for e in x if e.encode("ascii","ignore")])
#h =  g.apply(lambda x:" ".join(x)) #(using stemmed output as results in reduced duplic
#h
```

In [ ]:

```
#i = " ".join(w for w in h)
#i
```

In [116…

```
def clean_text(df):

    df["FactualNarrative"] = df["FactualNarrative"].apply(lambda x: x.strip())

    # Expand contractions
    df["FactualNarrative"] = df["FactualNarrative"].apply(lambda x: contractions.fix(x)

    # Tokenize
    df["tok"] = df["FactualNarrative"].apply(lambda x: word_tokenize(x))

    # Remove punctuation
```

```python
    punc = string.punctuation
    df["punct"] = df["tok"].apply(lambda x: [word for word in x if word not in punc])

    # Stopwords removal
    stop = ['January','February','March','April','May','June','July','August','Septembe
    df["stop"] = df["punct"].apply(lambda x: [w for w in x if w not in stop])

    # Remove numbers, except words that contain numbers.
    df["num"] = df["stop"].apply(lambda x: [n for n in x if not n.isnumeric()])

    # Remove non ascii characters
    df["ascii"] = df["num"].apply(lambda x: [e for e in x if e.encode("ascii","ignore")

    # Join the text into strings and join the strings into one text
    df["joined"] = df["ascii"].apply(lambda x:" ".join(x)) #(using stemmed output as re
    text = " ".join(e for e in df.joined)

    return text
```

In [117…

```python
# Get count of the words in the report text
df3["count"] = df3['FactualNarrative'].str.split().apply(len)
df3['count'].describe()
```

Out[117…

```
count    31003.000000
mean       685.914428
std        919.107312
min          3.000000
25%        175.000000
50%        344.000000
75%        784.000000
max      12730.000000
Name: count, dtype: float64
```

In [118…

```python
# How many words overall
df3['count'].sum()
```

Out[118…

```
21265405
```