VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

TITLE OF THE PROJECT

AIRPLANE CRASHES

A PROJECT REPORT

Submitted as a part of the course Exploratory Data Analysis

CSE3040

School of Computer Science And Engineering

VIT Chennai

FALL 2020-2021

TEAM MEMBERS

Manasa Reddy 19MIA1009

K.Harshitha 19MIA1026

M V Namitha 19MIA1031

Nagaruru Shreya 19MIA1044

Alaham Sanjana 19MIA1055

Course Faculty: **Dr. Subhra Patra**

ABSTRACT

The study examines the causes of crashes of aircrafts based on reported findings for the crash. The dataset used for this The study included data for all reported air crashes across the globe for the period from 1981 to 2019. The causes were classified into Various categories.

airplane crashes safety management is implemented through reactive, proactive, and predictive

methodologies. Unlike reactive and proactive safety, predictive safety can predict the next accident and enable prevention before an actual occurrence. The study outlined here

promotes predictive safety management through machine learning technologies using

large amounts of data to facilitate predictive modeling.

Multiple machine learning algorithms were used to identify the best for predicting the likely cause of accident based on features available. The Machine Learning Models used are K-Means Clustering Algorithm,DBSCAN clustering algorithm The goal was to Analyze and determine what model best predicts fatal and severe injury caused due to Airplane Crashes and further, what variables were most important in the prediction model.

INTRODUCTION

The introduction chapter for this study provides a project overview and lays a foundation for the follow-on chapters. The foundation begins with a concise foundation on general airplane accidents. Planes are being utilized as a method of transportation by a huge number of individuals consistently. The innovation of planes has done a ton of good as it is utilized by individuals for plenty of purposes, be it for going to outlandish spots , for conferences , or just to meet with one's family, individuals or companions.

The Wright brothers invented and flew the first airplane in 1903, recognized as "the first sustained and controlled heavier-than-air powered flight". The planes are getting well known as the methods for transportation among individuals constantly for the most part because of the way that they are the quickest method of transportation accessible. Even though airplanes have their own advantages, they do come with their fair share of disadvantages and problems. One of the cons that airplanes have is that they consume enormous amounts of fossil fuel which pose a great threat to environmental reserves of fossil fuels. The goal of accident analysis is to determine what can be done to prevent future mishaps.

More recent is the effort to move beyond proactive accident prevention to predictive methods empowered by AIML. Also the chances of surviving an aviation accident are almost close to zero. Likewise, planes are greatly influenced by natural conditions. It gets especially hard to control and navigate an airplane in bad weather. Strong winds cause turbulence, fog causes visual hindrance as well.

The most common reasons for aviation accidents are mentioned below :

1. Mechanical Faults(chance of equipment failing)

2. Pilot & Crew Error(most frequent causes of airplane wrecks, even a minor error lead to major failure)

3. Environmental Conditions(since they fly at high altitudes if weather deteriorates it is difficult to control and also if there is fog)

4. Air Traffic Control Errors(error made by air traffic controller while communicating with them or because of unheard)

5. Other Factors (like terrorist attacks, hijacking, medical issues of pilots etc..)

LITERATURE SURVEY/RELATED WORKS

Retired Chief Pilot from United Airlines Capt. Randy DeAngelis says that the pilots should be trained more efficiently so that if the aircraft has any problem such as adverse weather conditions they should be able to maintain control of the aircraft. The important point is to invest in Human Factor Research so that the risks due to human factors will be reduced significantly.

(Source :
http://www.arnellent.com/aviation-crew-RandyDeAngelis.html)
Retired Chief Pilot from United Airlines Capt. Randy DeAngelis says that the pilots should be trained more efficiently so that if the aircraft has any problem such as adverse weather conditions they should be able to maintain control of the aircraft. The important point is to invest in Human Factor Research so that the risks due to human factors will be reduced significantly.

(Source :
http://www.arnellent.com/aviation-crew-RandyDeAngelis.html)

Stephens and Ukpere (2014) in their paper say that on examination of causes of air crashes over time has shown that during the time when airplane travel was still in infancy, most of the crashes were due to fuel starvation, some flaws in the aircraft design and lightning.

EXISTING WORK

Dataset used:

The dataset that we have used for this project is from Kaggle.

The dataset contains 5268 observations(crashes) and 13 features namely Date, Time, Location, Operator, Flight, Route, Type, Registration, Aboard, Fatalities, Summary, Ground, cn/in.

The dataset contains data of airplane accidents involving civil, commercial and military transport worldwide

Methodology :

Design of the study:

-Discover which regions are more prone to air crashes

-Examine the causes of accidents

-Analyse the different air crashes based on regions

Area of study:

This study covers worldwide occurrences of air crashes that happened since 1908.

Method Of Data Collection:

Research data was collected from kaggle.

Data Analytical Tool:

To analyse the data we used kaggle as a tool. The result are tabulated in a simple way for easy understanding and comprehension

Code

Importing the Libraries:

We imported libraries such as numpy , pandas, seaborn , matplotlib. Then we Imported the Data into a Pandas DataFrame for further analysis

Data types and values:

We found out the datatypes of all attributes and missing values of each of the attributes

The highest missing values were in Route so we dropped the attribute.

Birds-eye Approach:

Then we used Birds-eye approach for the observations

It is always good to start from a "higher-view" and then gradually move into deeper analysis. For the sake of simplicity and making our analysis fruitful, let us define two characteristics of the the data -

1. Dimensions: are variables in the context of analysis
2. Measures: the change in these values, changes the "measure"

We can observe that this approach is similar to the coordinate geometry analogy (dimensions and values at a coordinate). We can "measure" 2 things from the data :

1. Crashes
2. Fatalities

Now these measures can be a function of one or many dimensions. In the first part of the analysis, we will look at the following dimensions individually:

1. Is it Military?
2. Country
3. Operator
4. Type
5. Route

In the first part, we explore basic questions such as: Is there a pattern between Military flights and fatalities?. This way we start off from a birds-eye view and move towards deeper patterns between different dimensions.

In the higher levels, we can explore the relationship between groups of dimensions against the measures. In higher levels, we can answer questions/ hypotheses such as : Is there a pattern between Military planes and Operators in terms of fatalities per crash?

Creating "Country" column from "Location" column:

Data analysis always involves wrangling and churning existing data to give a new meaningful perspective.

We can separate the Location by "," and have the last entry as country.

Here, In the dataset we also observe that we have states from the USA - we can group them as one later.

- Extracting information about year of crash

Visualations:

Then we made some visualizations:

1.Total Airplane Crashes per year

2.Total of people aboard airplanes per year

3.Survivors vs Fatalities vs Killed on ground per year

PROPOSED WORK

Text Analysis with K-Means clustering

K means Clustering is an unsupervised machine learning algorithm that aims to partition observations into clusters in which each observation belongs to the cluster with the nearest mean.The algorithm we followed in the code is

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be different from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means assign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

CODE:

Importing needed modules :

We imported the needed modules for k-means clustering such as sklearn.cluster, sklearn.metrics, sklearn.preprocessing, seaborn,matplotlib, sklearn.decomposition, sklearn.feature_extraction.

Data preparation :

In the 'Summary' column, we have NaN values as well, so we're going to create a new dataframe with the 'Summary' data and dropping all rows with NaN values

Model fitting :

And now we fit the model. For this analysis, we'll be using the KMeans algorithm with 5 clusters.

Visualization :

As we know the dimension of features that we obtained from TfIdfVectorizer is quite large we need to reduce the dimension before we can plot. For this, we'll use PCA to transform our high dimensional features into 2 dimensions.

Prediction :

Then we predicted the number of crashes due to engine failure and terrorism.

Text Clustering with DBSCAN

Density-Based Clustering refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

How does the DBSCAN algorithm work?

Let X = {x1, x2, x3, ..., xn} be the set of data points. DBSCAN requires two parameters: ε (eps) and the minimum number of points required to form a cluster (minPts).

1) Start with an arbitrary starting point that has not been visited.

2) Extract the neighborhood of this point using ε (All points which are within the ε distance are neighborhoods).

3) If there are sufficient neighborhoods around this point then the clustering process starts and the point is marked as visited else this point is labeled as noise (Later this point can become the part of the cluster).

4) If a point is found to be a part of the cluster then its ε neighborhood is also the part of the cluster and the above procedure from step 2 is repeated for all ε neighborhood points. This is repeated until all points in the cluster are determined.

5) A new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.

6) This process continues until all points are marked as visited.

Importing needed modules:

We imported modules needed for DBscan such as sklearn.preprocessing ,sklearn.decomposition , sklearn.cluster

Vectorizing:

Then we normalized summary attribute to english

Plotting the graph for DB scan:

We grouped them into 12 labels and plotted them in a scatter plot.

CAUSES OF AIRPLANE CRASHES:

While we were analysing the words in each topic, we have identified what are the causes for the airplane crashes

FLOWCHART:

TASK 1: Extracting data set

TASK 2: Pre processing of data set

TASK 3 :Outlier detection

TASK 4 : Visualizations

TASK 5 :Review 2

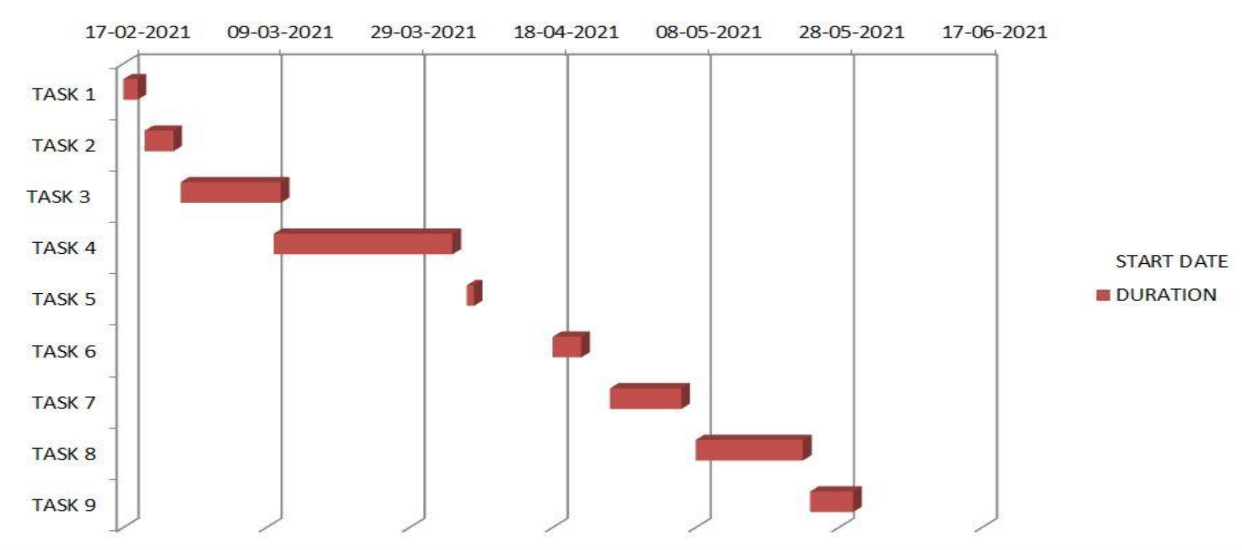
TASK 6: Model selection

TASK 7:k Means Clustering

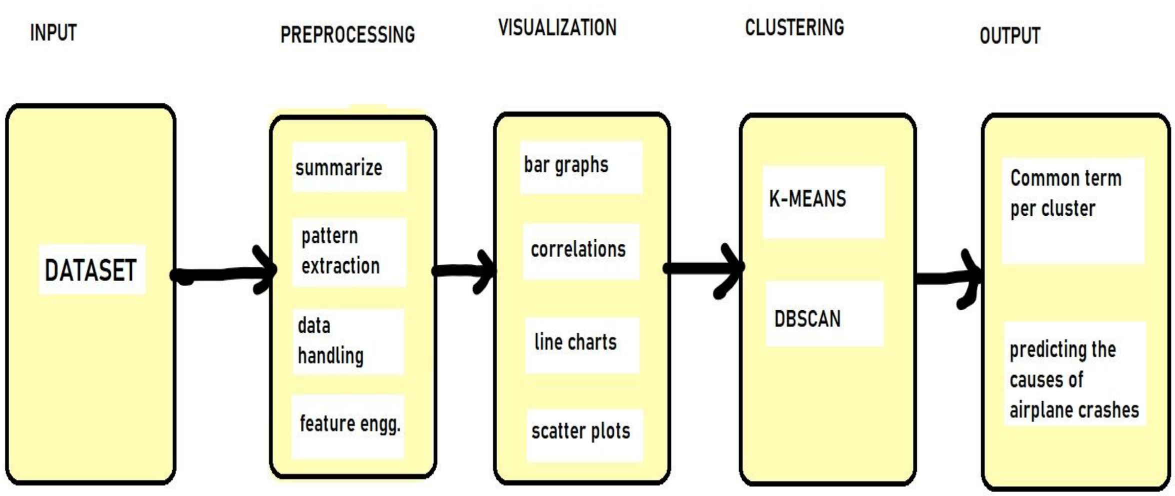
TASK 8: DB scan clustering

TASK 9 : Final report preparation

GANTT CHART



FLOWCHART



WORKING MODULES:

K-Means clustering:

K means Clustering is an unsupervised machine learning algorithm that aims to partition observations into clusters in which each observation belongs to the cluster with the nearest mean.The algorithm we followed in the code is

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be different from the input dataset).

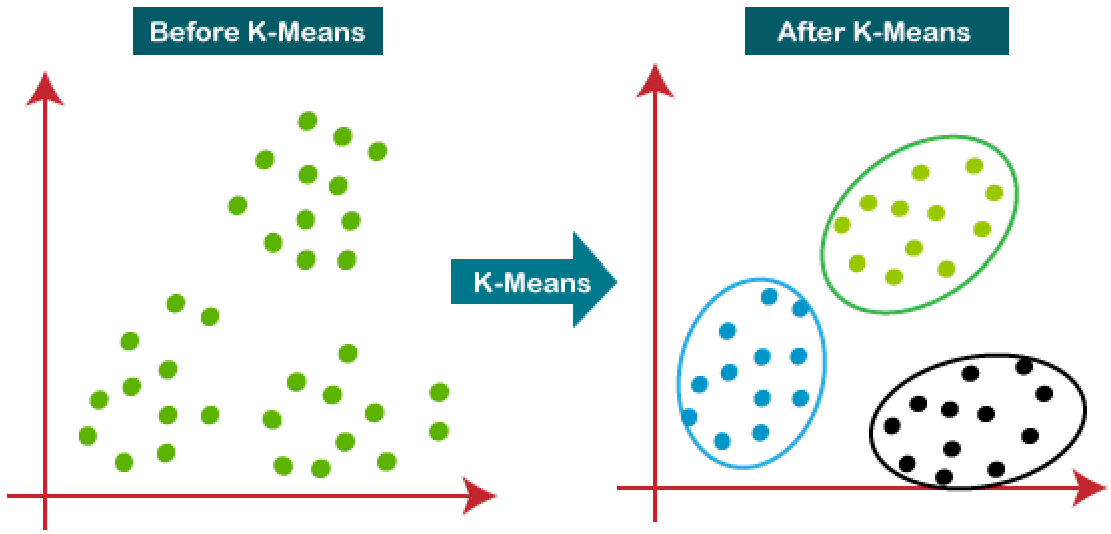
Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means assign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.



DBscan clustering:

Density-Based Clustering refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

Steps followed:

1) Start with an arbitrary starting point that has not been visited.

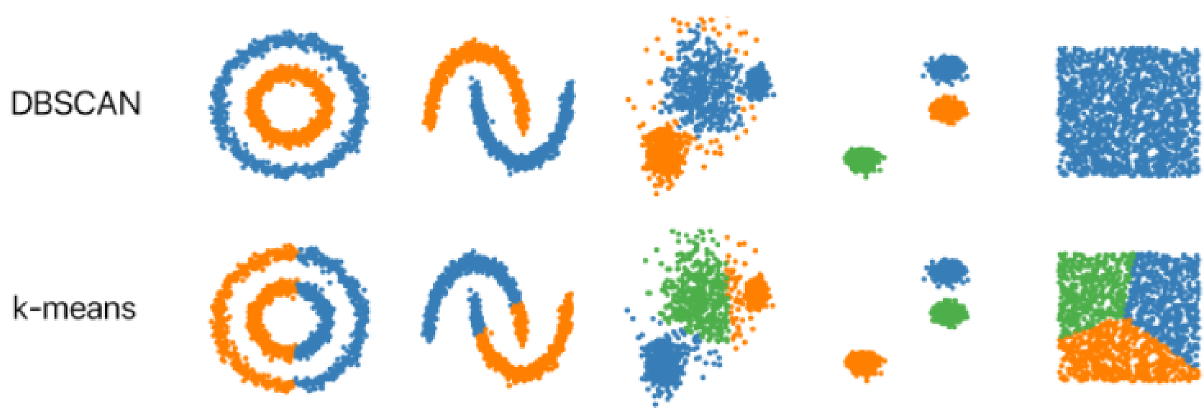
2) Extract the neighborhood of this point using ε (All points which are within the ε distance are neighborhoods).

3) If there are sufficient neighborhoods around this point then the clustering process starts and the point is marked as visited else this point is labeled as noise (Later this point can become the part of the cluster).

4) If a point is found to be a part of the cluster then its ε neighborhood is also the part of the cluster and the above procedure from step 2 is repeated for all ε neighborhood points. This is repeated until all points in the cluster are determined.

5) A new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.

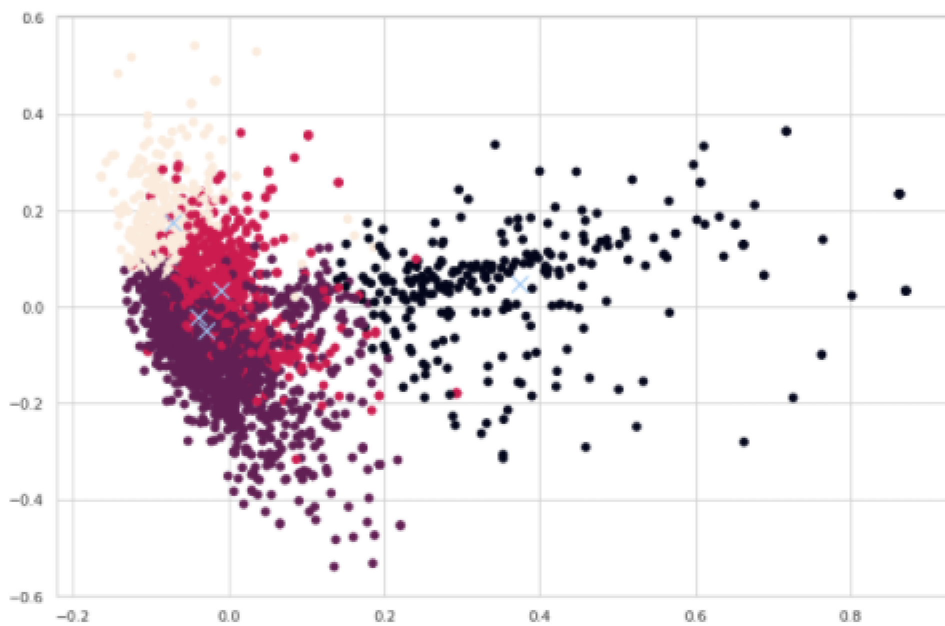
6) This process continues until all points are marked as visited.



RESULT AND ANALYSIS

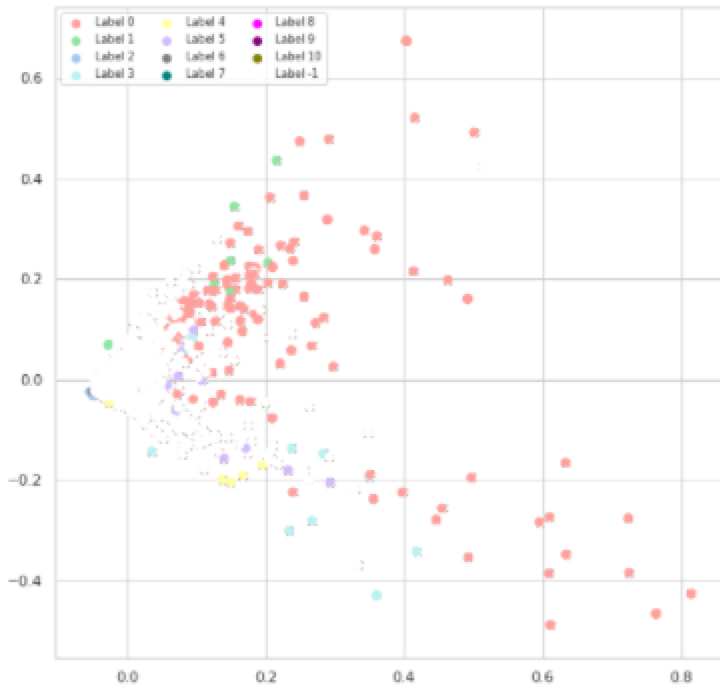
K Means clustering:

By analysing the graphs we came to the conclusion that the number of clustering can be considered as k=4. And after clustering we have found out that the group each point belongs to clusters.



DBscan clustering:

For DBscan, as there are a sufficient number of points within this neighborhood, we started the clustering process and the current data point is termed the first point in the new cluster. Otherwise, the point is labeled as noise. Once we're done with the current cluster, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise. This process has been repeated until all points are marked as visited. At the end, each point has been marked as either belonging to a cluster or being noisy.



Using these machine learning methods we analysed few causes for frequent airplane crashes

Cause 1: Spatial disorientation due to bad weather conditions.

Cause 2: Stalled the engine. The explosion or destruction of the aircraft from falling to the ground or collision with the building.

Cause 3: Failure of the rotor or problems with the fuselage (specifically problems with the tail). It is also a possible mistake of the Air Traffic Control centre.

Cause 4: Bad weather conditions: strong wind, snow, ice. The plane disappeared from radar.

Cause 5: Taking off without clearance from ATC. ATC or pilot error.

Cause 6: Crash due to manoeuvring. Most likely refers to the testing and training missions.

Cause 7: The plane was hijacked or captured by the rebels. Fell to the ground due to issues with piloting or bad weather conditions.

Cause 8: The plane was destroyed by explosion and destruction of the fuselage. The cause of the explosion could be a bomb or fuel tank.

Cause 9: The malfunction of the autopilot and remote control systems. Most likely associated with transport aircraft.

Cause 10: Navigation problems, technical malfunction.

Results:

- Majority of crash sites appear in the northern hemisphere.
- Majority of crashes occur near the coast of countries or continents.
- More than 2/3 of large passenger crashes have high fatality
- No clear location advantage for high survival vs. high fatality other than more survivable crashes occur over land and more high fatality crashes occur over oceans or seas.

SCREENSHOTS OF CODE AND OUTPUTS FOR EXISTING WORK

1. Importing modules

```
[ ]  #Setting up the environemnt

     import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib
     import re
     import matplotlib.pyplot as plt
     import scipy.stats as stats
     from datetime import date, timedelta, datetime
     from collections import Counter
     import os

     print(os.listdir("../input/"))
```

```
['Airplane_Crashes_and_Fatalities_Since_1908.csv']
```

2. Loading the data

```
[ ]  data = pd.read_csv("../input/Airplane_Crashes_and_Fatalities_Since_1908.csv")
```

```
[ ]  np.random.seed(42)
     obs, feat = data.shape
     data.sample(5)
```

	Date	Time	Location	Operator	Flight #	Route	Type	Registration	cn/In	Aboard	Fatalities	Ground	Summary
4793	07/30/2001	16:00	Haines, Alaska	Air Taxi	NaN	Shagway, AK - Return	Piper PA-32-300	N39586	32-7840168	6.0	6.0	0.0	The sightseeing plane impacted the side of a m...
5020	11/28/2004	09:55	Montrose, Colorado	Glow Air/Air Castle - Charter	73	Montrose, CO - South Bend, IN	Canadair CL-601-2A12 Challenger	N873G	3009	6.0	3.0	0.0	While attempting to take off from R31, the air...
655	06/20/1944	NaN	Porto Alegre, Brazil	Varig	NaN	NaN	Lockheed 10C Electra	PP-VAG	1008	10.0	10.0	0.0	Crashed into a river.
1874	10/09/1963	NaN	Near Marseilles, France	Aeronaves de Panama	NaN	Marseilles, France - Dhah, Saudi Arabia	Douglas C-74	HP-385	13915	6.0	6.0	0.0	Crashed shortly after taking off with a cargo ...
2318	06/05/1969	07:12	North Bend, Oregon	Eureka Aero Inc. - Air Taxi	NaN	NaN	Cessna 337C	N2665S	NaN	3.0	3.0	0.0	Collided with trees on approach. Improper IFR ...

```
[ ] data['Survived'] = data['Aboard'] - data['Fatalities']
    data['Survived'].fillna(0, inplace = True)
```

To ensure our data file read correctly, we see first five columns

```
[ ] data.head()
```

	Date	Time	Location	Operator	Flight #	Route	Type	Registration	cn/In	Aboard	Fatalities	Ground	Summary	Survived
0	09/17/1908	17:18	Fort Myer, Virginia	Military - U.S. Army	NaN	Demonstration	Wright Flyer III	NaN	1	2.0	1.0	0.0	During a demonstration flight, a U.S. Army fly...	1.0
1	07/12/1912	06:30	AtlantiCity, New Jersey	Military - U.S. Navy	NaN	Test flight	Dirigible	NaN	NaN	5.0	5.0	0.0	First U.S. dirigible Akron exploded just offsh...	0.0
2	08/06/1913	NaN	Victoria, British Columbia, Canada	Private	-	NaN	Curtiss seaplane	NaN	NaN	1.0	1.0	0.0	The first fatal airplane accident in Canada oc...	0.0

3. Data Info and Manipulation

```
[ ]  data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5268 entries, 0 to 5267
Data columns (total 14 columns):
Date            5268 non-null object
Time            3049 non-null object
Location        5248 non-null object
Operator        5250 non-null object
Flight #        1069 non-null object
Route           3562 non-null object
Type            5241 non-null object
Registration    4933 non-null object
cn/In           4040 non-null object
Aboard          5246 non-null float64
Fatalities      5256 non-null float64
Ground          5246 non-null float64
Summary         4878 non-null object
Survived        5268 non-null float64
dtypes: float64(4), object(10)
memory usage: 576.3+ KB
```

```
[ ] data.shape
```

```
(5268, 14)
```

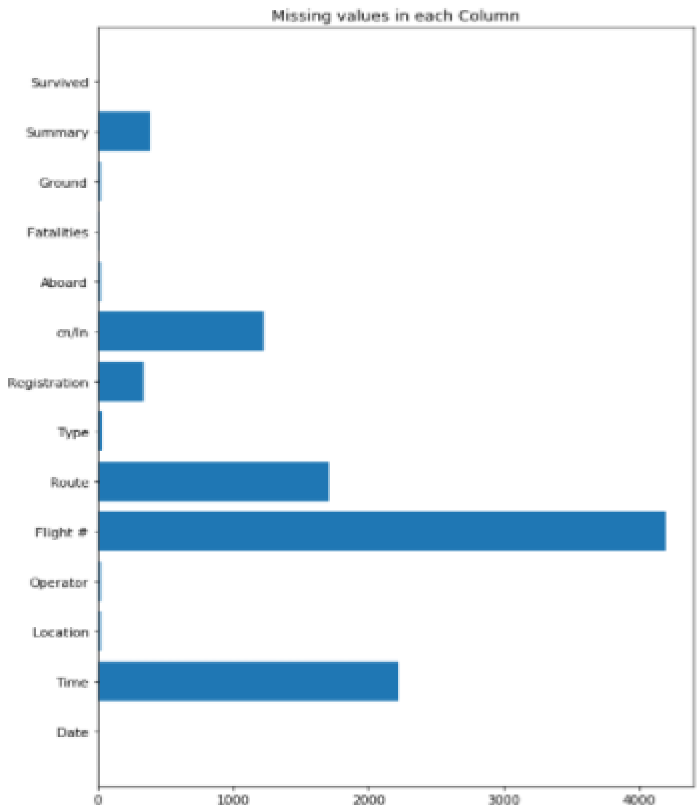
```
[ ] data.describe()
```

	Aboard	Fatalities	Ground	Survived
count	5246.000000	5256.000000	5246.000000	5268.000000
mean	27.554518	20.068303	1.608845	7.439825
std	43.076711	33.199952	53.987827	28.089951
min	0.000000	0.000000	0.000000	0.000000
25%	5.000000	3.000000	0.000000	0.000000
50%	13.000000	9.000000	0.000000	0.000000
75%	30.000000	23.000000	0.000000	2.000000
max	644.000000	583.000000	2750.000000	516.000000

```
[ ] def null_table(data):
        print(pd.isnull(data).sum())

    null_table(data)
```

```
Date               0
Time            2219
Location          20
Operator          18
Flight #        4199
Route           1706
Type              27
Registration     335
cn/In           1228
Aboard            22
Fatalities        12
Ground            22
Summary          390
Survived           0
dtype: int64
```

```
[ ] a = [(i, data[i].isna().sum()) for i in data.columns]
    labels, ys = zip(*a)

    plt.figure(figsize=[8,12])
    plt.barh(labels, ys , height=0.8)
    plt.title("Missing values in each Column")

    plt.rcParams.update({'font.size': 20})
```




```
[ ] data['Fatalities'].fillna(0, inplace = True)
    data['Aboard'].fillna(0, inplace = True)
    data['Ground'].fillna(0, inplace = True)
```

With data on number of fatalities and people aboard, we create a new variable with the number of people that survived the crash and call this variable 'Survived'. We also replace any NaN value with 0 on this column.

```
[ ] data['Survived'] = data['Aboard'] - data['Fatalities'] - data['Ground']
    data['Has_Survivors'] = 1
    data.loc[data['Survived'] == 0, 'Has_Survivors'] = 0
```

Now our dataframe looks like this.

```
[ ] data.head()
```

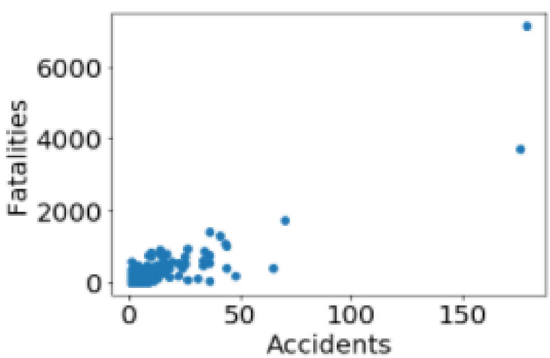
	Date	Time	Location	Operator	Flight #	Route	Type	Registration	cn/In	Aboard	Fatalities	Ground	Summary	Survived
0	09/17/1908	17:18	Fort Myer, Virginia	Military - U.S. Army	NaN	Demonstration	Wright Flyer III	NaN	1	2.0	1.0	0.0	During a demonstration flight, a U.S. Army fly...	1.0
1	07/12/1912	06:30	AtlantiCity, New Jersey	Military - U.S. Navy	NaN	Test flight	Dirigible	NaN	NaN	5.0	5.0	0.0	First U.S. dirigible Akron exploded just offsh...	0.0
2	08/06/1913	NaN	Victoria, British Columbia, Canada	Private	-	NaN	Curtiss seaplane	NaN	NaN	1.0	1.0	0.0	The first fatal airplane accident in Canada oc...	0.0
3	09/09/1913	18:30	Over the North Sea	Military - German Navy	NaN	NaN	Zeppelin L-1 (airship)	NaN	NaN	20.0	14.0	0.0	The airship flew into a thunderstorm and encou...	6.0
4	10/17/1913	10:30	Near Johannisthal, Germany	Military - German Navy	NaN	NaN	Zeppelin L-2 (airship)	NaN	NaN	30.0	30.0	0.0	Hydrogen gas which was being vented was sucked...	0.0

```
[ ] data_first = data.copy()
    data['Time'] = data ['Time'].replace(np.nan, '00:00') ####
    data['Time'] = data ['Time'].str.replace('c: ', '')
    data['Time'] = data ['Time'].str.replace('c:', '')
    data['Time'] = data ['Time'].str.replace('c', '')
    data['Time'] = data ['Time'].str.replace('12\'20', '12:20')
    data['Time'] = data ['Time'].str.replace('18.40', '18:40')
    data['Time'] = data ['Time'].str.replace('0943', '09:43')
    data['Time'] = data ['Time'].str.replace('22\'08', '22:08')
    data['Time'] = data ['Time'].str.replace('114:20', '00:00')
```

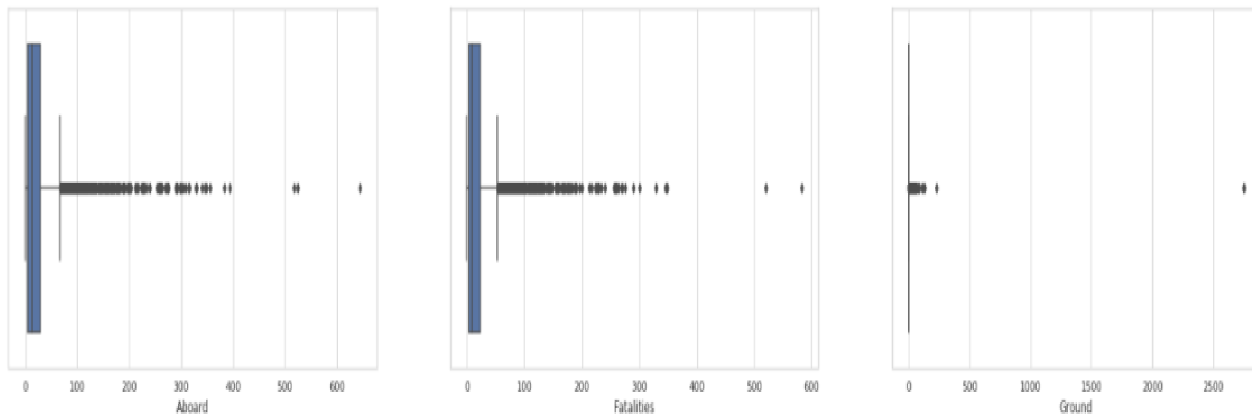
```python
data_first = data.copy()
data['Time'] = data ['Time'].replace(np.nan, '00:00') ####
data['Time'] = data ['Time'].str.replace('c: ', '')
data['Time'] = data ['Time'].str.replace('c:', '')
data['Time'] = data ['Time'].str.replace('c', '')
data['Time'] = data ['Time'].str.replace('12\'20', '12:20')
data['Time'] = data ['Time'].str.replace('18.40', '18:40')
data['Time'] = data ['Time'].str.replace('0943', '09:43')
data['Time'] = data ['Time'].str.replace('22\'08', '22:08')
data['Time'] = data ['Time'].str.replace('114:20', '00:00')

data.Operator = data.Operator.str.upper() #just to avoid duplicates like 'British Airlines' and 'BRITISH Airlines'
#data['Fatalities'] = data['Fatalities'].fillna(0)
operator = data[['Operator','Fatalities']].groupby('Operator').agg(['sum','count'])
```

4. Outlier Detection

```python
data['Fatalities'] = data['Fatalities'].fillna(0)
X = operator['Fatalities','count']
Y = operator['Fatalities','sum']
plt.scatter(X, Y,label='Operators')
plt.ylabel('Fatalities')
plt.xlabel('Accidents');
```



```
[ ] plt.figure(figsize=(30, 5))
    ind = 1

    for col in data.loc[:,'Aboard':'Ground'].columns:
        plt.subplot(1, 3, ind)
        sns.boxplot(x=data[col])
        ind += 1
```



```
[ ] data_first['Fatalities_percentage'] = data['Fatalities'] / data['Aboard']
    print(data_first['Fatalities_percentage'].head(5))
```

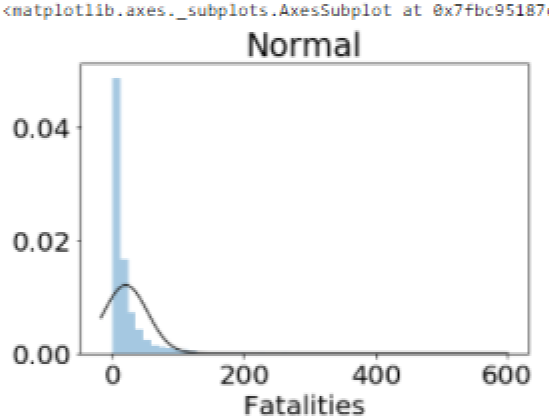
```
0    0.5
1    1.0
2    1.0
3    0.7
4    1.0
Name: Fatalities_percentage, dtype: float64
```

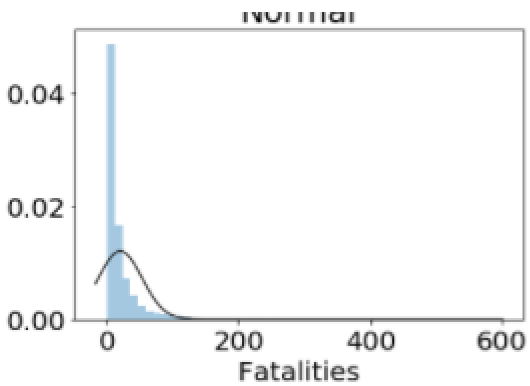
```
[ ] print("Skewness: %f" % data['Fatalities'].skew())
    print("Kurtosis: %f" % data['Fatalities'].kurt())
```

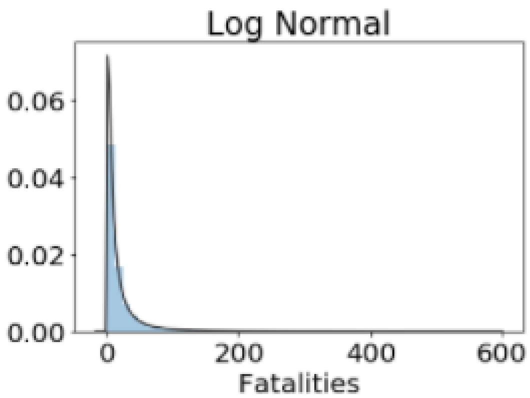
```
Skewness: 4.952818
Kurtosis: 42.889113
```

```
y = data['Fatalities']
plt.figure(2); plt.title('Normal')
sns.distplot(y, kde=False, fit=stats.norm)
plt.figure(3); plt.title('Log Normal')
sns.distplot(y, kde=False, fit=stats.lognorm)
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc95187c18>
```





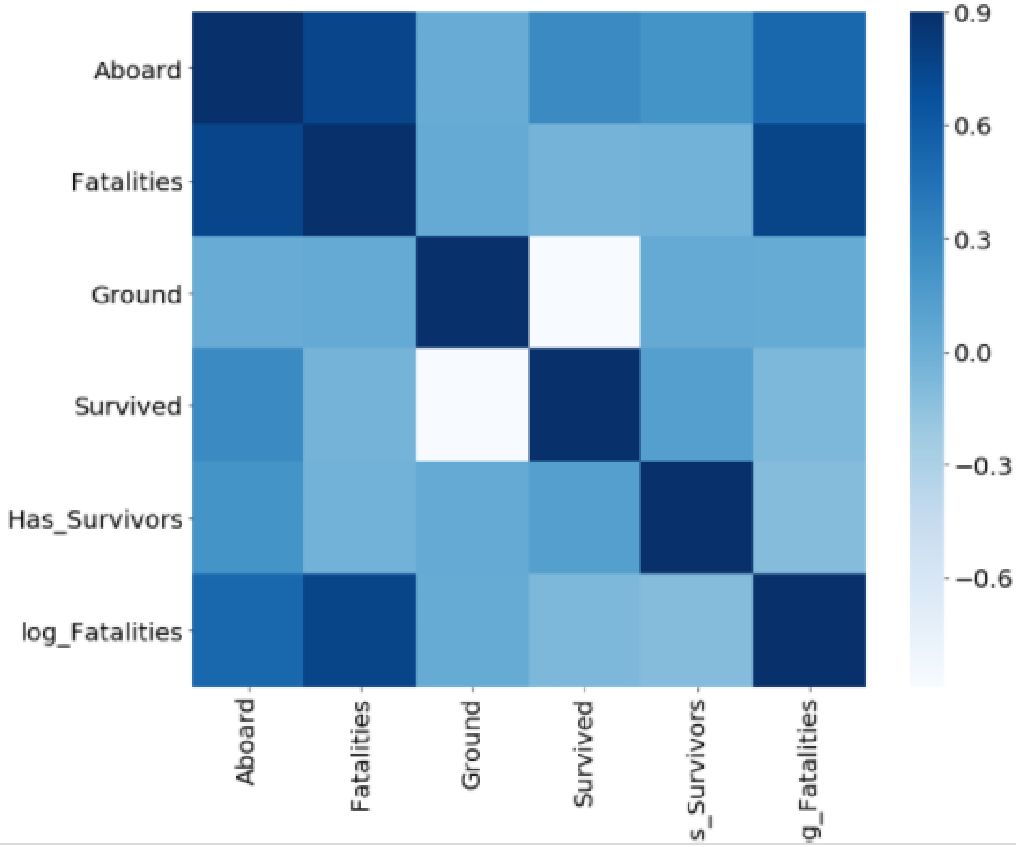



```
[ ] data["log_Fatalities"] = np.log1p(data["Fatalities"])
    print("Skewness: %f" % data['log_Fatalities'].skew())
    print("Kurtosis: %f" % data['log_Fatalities'].kurt())
```

```
Skewness: 0.339361
Kurtosis: -0.467497
```

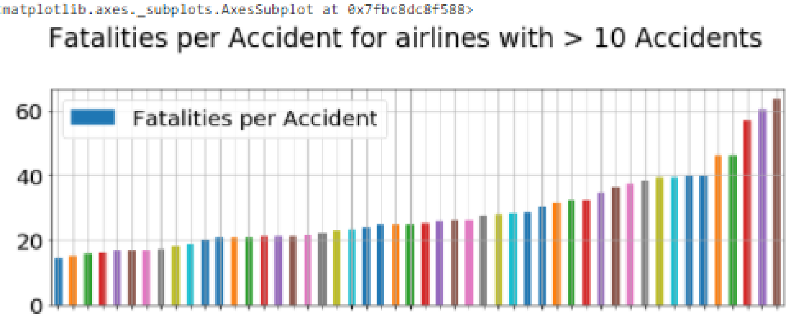
```
[ ] #Correlation matrix
    corr = data.corr()
    plt.subplots(figsize=(13,10))
    sns.heatmap(corr, vmax=0.9, cmap="Blues", square=True)
```

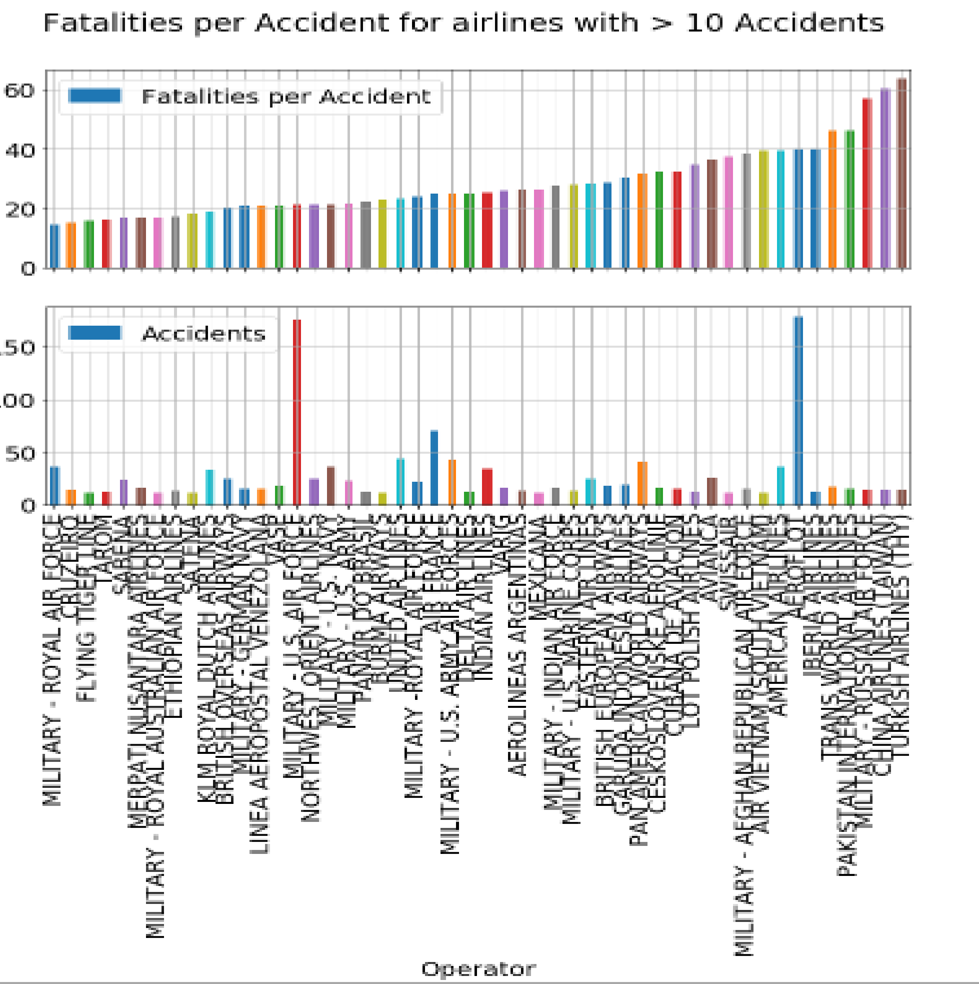
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc8ddb72e8>
```



```
matplotlib.rcParams['figure.figsize'] = (12.0, 8.0)
#Lets take a look at the proportion of fatalities per accident for specific operators.
#This bears out some interesting statistics. Thanks for the suggestion @Jos Smit.
props = operator['Fatalities'].reset_index()
props['Fatalities per Accident'] = props['sum']/props['count']
props.columns = ['Operator','Fatalities','Accidents','Fatalities per Accident']

fig_p,(axp1,axp2) = plt.subplots(2,1,sharex = True)
minacc = 10
fig_p.suptitle('Fatalities per Accident for airlines with > %s Accidents' % minacc)
propstoplot = props[props['Accidents']>minacc]
propstoplot.sort_values('Fatalities per Accident').tail(50).plot(x = 'Operator'
                                                                , y = 'Fatalities per Accident'
                                                                , ax = axp1
                                                                , kind = 'bar'
                                                                , grid = True)
propstoplot.sort_values('Fatalities per Accident').tail(50).plot(x = 'Operator'
                                                                , y = 'Accidents'
                                                                , ax = axp2
                                                                , kind = 'bar'
                                                                , grid = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc8dc8f588>
```




Fatalities per Accident for airlines with > 10 Accidents
Fatalities per Accident
Accidents
Operator

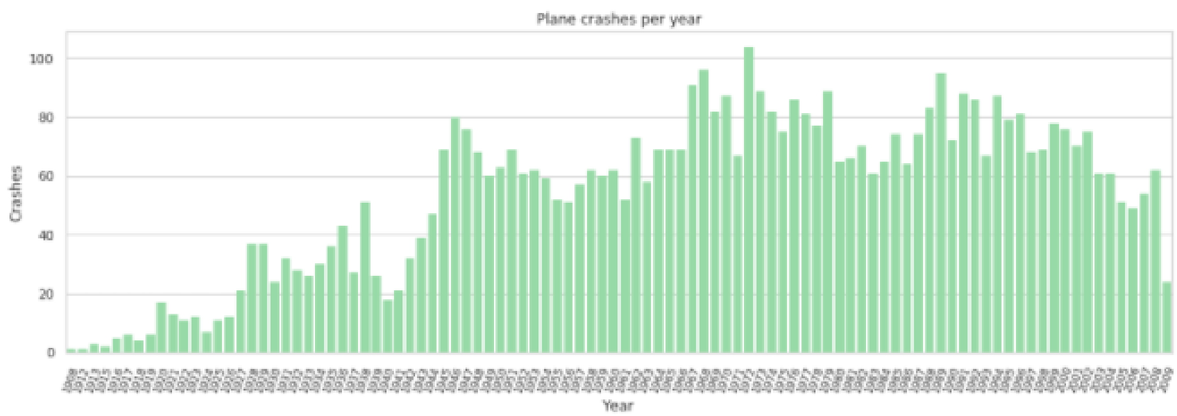
```
[ ] data['Date'] = pd.to_datetime(data['Date'])
    data['Day'] = data['Date'].map(lambda x: x.day)
    data['Year'] = data['Date'].map(lambda x: x.year)
    data['Month'] = data['Date'].map(lambda x: x.month)
```

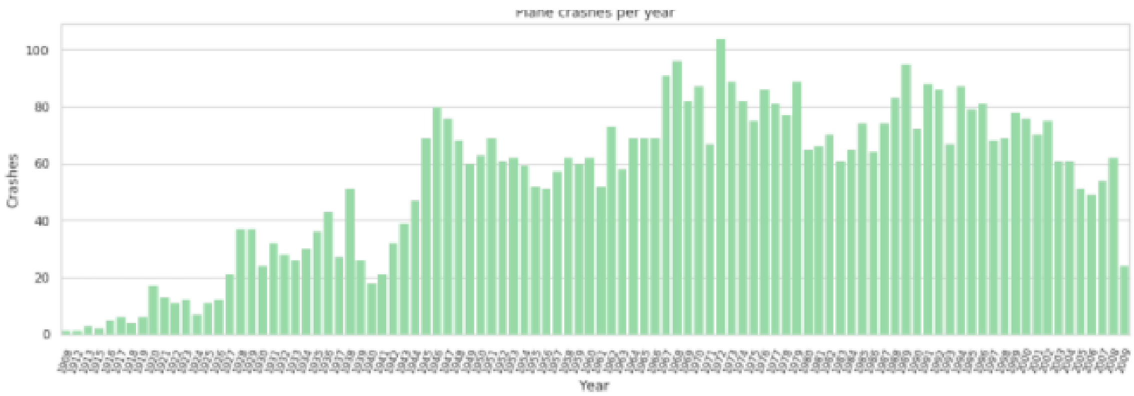
```
[ ] crashes_per_year = Counter(data['Year'])
    years = list(crashes_per_year.keys())
    crashes_year = list(crashes_per_year.values())
    crashes_per_day = Counter(data['Day'])
    days = list(crashes_per_day.keys())
    crashes_day = list(crashes_per_day.values())
```

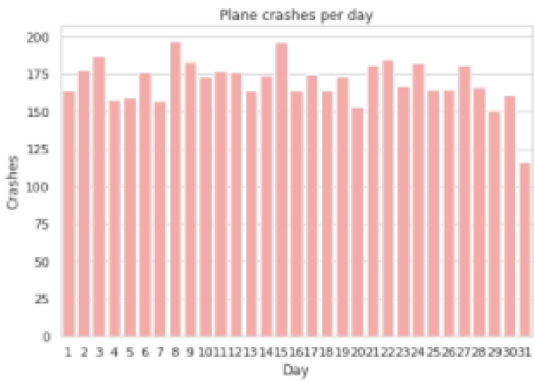
```
[ ] def get_season(month):
        if month >= 3 and month <= 5:
            return 'spring'
        elif month >= 6 and month <= 8:
            return 'summer'
        elif month >= 9 and month <= 11:
            return 'autumn'
        else:
            return 'winter'

    data['Season'] = data['Month'].apply(get_season)
```

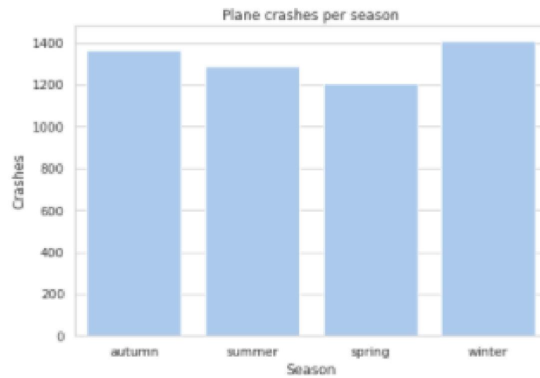
```
[ ] crashes_per_season = Counter(data['Season'])
    seasons = list(crashes_per_season.keys())
    crashes_season = list(crashes_per_season.values())

    sns.set(style="whitegrid")
    sns.set_color_codes("pastel")

    fig = plt.figure(figsize=(14, 10))

    sub1 = fig.add_subplot(211)
    sns.barplot(x=years, y=crashes_year, color='g', ax=sub1)
    sub1.set(ylabel="Crashes", xlabel="Year", title="Plane crashes per year")
    plt.setp(sub1.patches, linewidth=0)
    plt.setp(sub1.get_xticklabels(), rotation=70, fontsize=9)
```

```
sns.set(style="whitegrid")
sns.set_color_codes("pastel")

fig = plt.figure(figsize=(14, 10))

sub1 = fig.add_subplot(211)
sns.barplot(x=years, y=crashes_year, color='g', ax=sub1)
sub1.set(ylabel="Crashes", xlabel="Year", title="Plane crashes per year")
plt.setp(sub1.patches, linewidth=0)
plt.setp(sub1.get_xticklabels(), rotation=70, fontsize=9)

sub2 = fig.add_subplot(223)
sns.barplot(x=days, y=crashes_day, color='r', ax=sub2)
sub2.set(ylabel="Crashes", xlabel="Day", title="Plane crashes per day")

sub3 = fig.add_subplot(224)
sns.barplot(x=seasons, y=crashes_season, color='b', ax=sub3)
texts = sub3.set(ylabel="Crashes", xlabel="Season", title="Plane crashes per season")

plt.tight_layout(w_pad=4, h_pad=3)
```

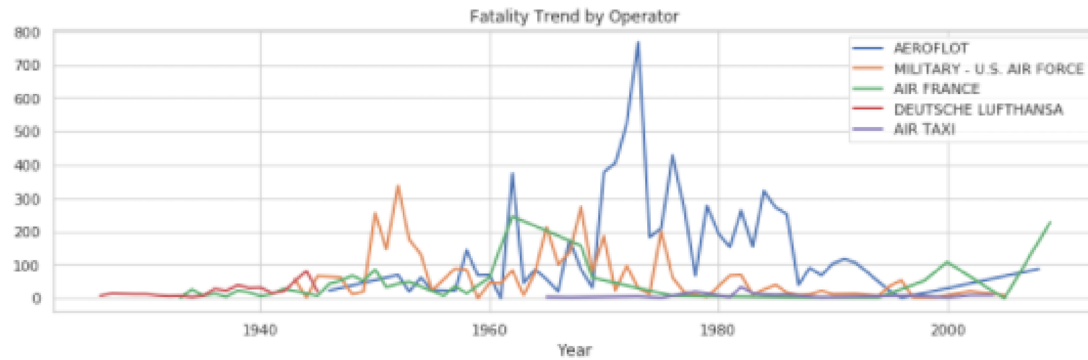


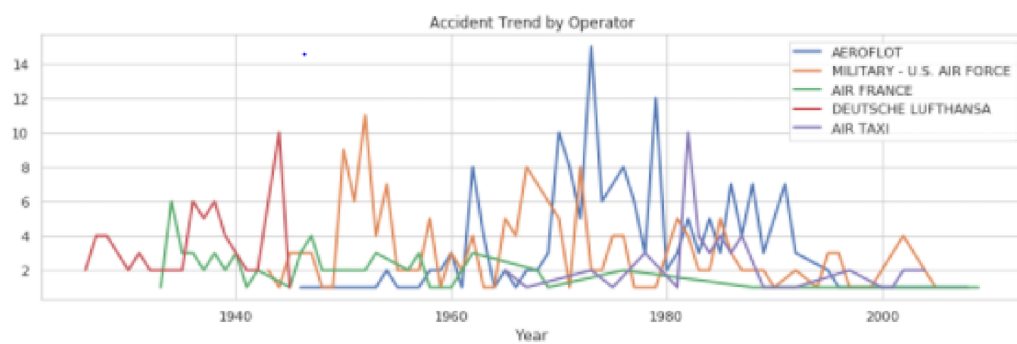

Plane crashes per year
Crashes
0
20
40
60
80
100
Year


Plane crashes per day
Crashes
0
25
50
75
100
125
150
175
200
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Day


Plane crashes per season
Crashes
0
200
400
600
800
1000
1200
1400
autumn
summer
spring
winter
Season

```
[ ] accidents = operator['Fatalities','count'].sort_values(ascending=False)
    interestingOps = accidents.index.values.tolist()[0:5]
    optrend = data[['Operator','Year','Fatalities']].groupby(['Operator','Year']).agg(['sum','count'])
    ops = optrend['Fatalities'].reset_index()
    fig,axtrend = plt.subplots(2,1)
    for op in interestingOps:
        ops[ops['Operator']==op].plot(x='Year',y='sum',ax=axtrend[0],grid=True,linewidth=2)
        ops[ops['Operator']==op].plot(x='Year',y='count',ax=axtrend[1],grid=True,linewidth=2)

    axtrend[0].set_title('Fatality Trend by Operator')
    axtrend[1].set_title('Accident Trend by Operator')
    linesF, labelsF = axtrend[0].get_legend_handles_labels()
    linesA, labelsA = axtrend[1].get_legend_handles_labels()
    axtrend[0].legend(linesF,interestingOps)
    axtrend[1].legend(linesA,interestingOps)
    plt.tight_layout()
```





Summary of the count of accidents per year:

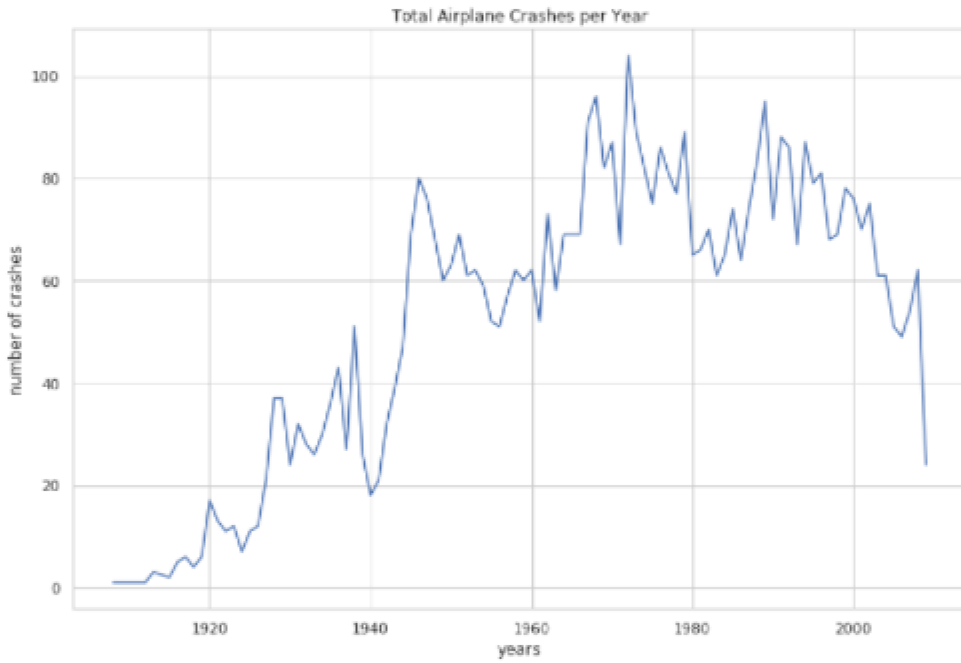
```
[ ] total_crashes_year = data[['Year', 'Date']].groupby('Year').count()
    total_crashes_year = total_crashes_year.reset_index()
    total_crashes_year.columns = ['Year', 'Crashes']
```

Line plot with Seaborn.

```
[ ] sns.lineplot(x = 'Year', y = 'Crashes', data = total_crashes_year)
    plt.title('Total Airplane Crashes per Year')
    plt.xlabel('years')
    plt.ylabel('number of crashes')

    Text(0,0.5,'number of crashes')
```

```
[ ] sns.lineplot(x = 'Year', y = 'Crashes', data = total_crashes_year)
    plt.title('Total Airplane Crashes per Year')
    plt.xlabel('years')
    plt.ylabel('number of crashes')
```

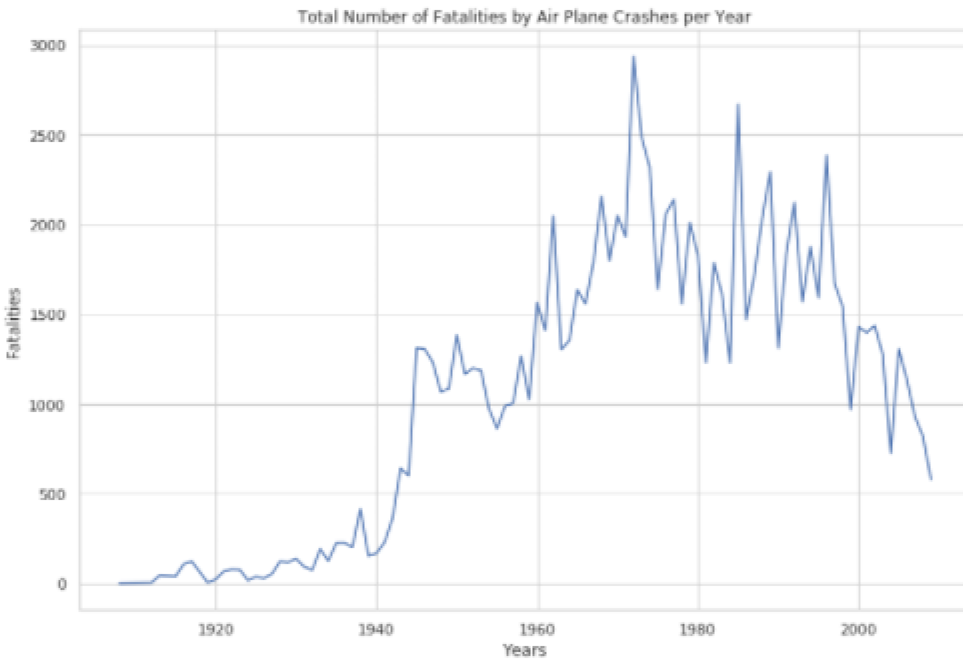
```
Text(0,0.5,'number of crashes')
```



```
[ ] pcdeaths_year = data[['Year', 'Fatalities']].groupby('Year').sum()
    pcdeaths_year.reset_index(inplace = True)
```

```
[ ] # Plot
    sns.lineplot(x = 'Year', y = 'Fatalities', data = pcdeaths_year)
    plt.title('Total Number of Fatalities by Air Plane Crashes per Year')
    plt.xlabel('Fatalities')
    plt.xlabel('Years')
```

```
Text(0.5,0,'Years')
```

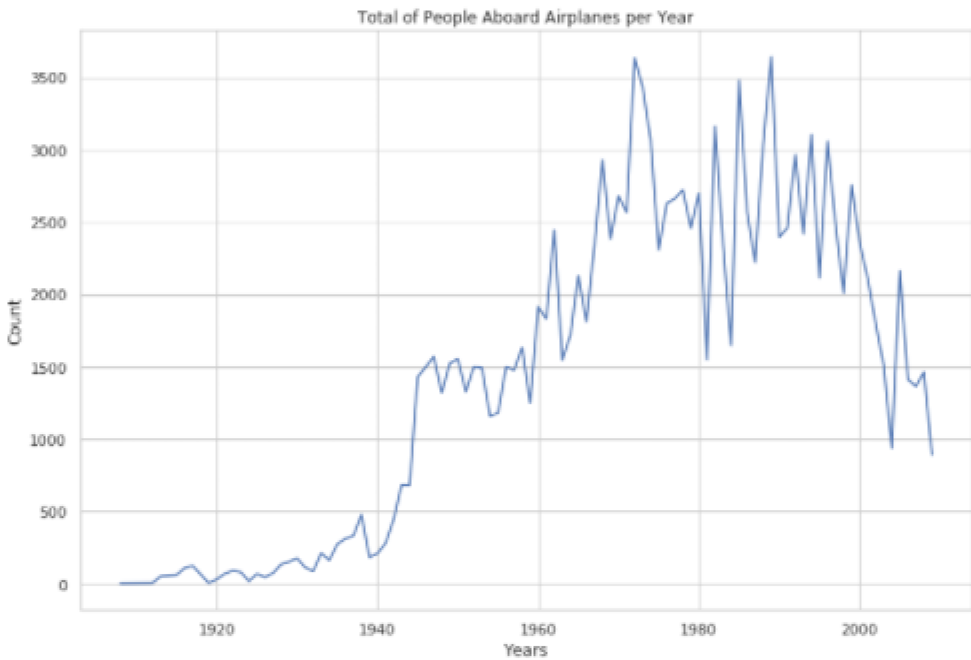


Plot of the people aboard in airplanes per year:

```
[ ] # summarise
    abrd_per_year = data[['Year', 'Aboard']].groupby('Year').sum()
    abrd_per_year = abrd_per_year.reset_index()
```

```
[ ] # plot
    sns.lineplot(x = 'Year', y = 'Aboard', data = abrd_per_year)
    plt.title('Total of People Aboard Airplanes per Year')
    plt.xlabel('Years')
    plt.ylabel('Count')
```

```
Text(0,0.5,'Count')
```



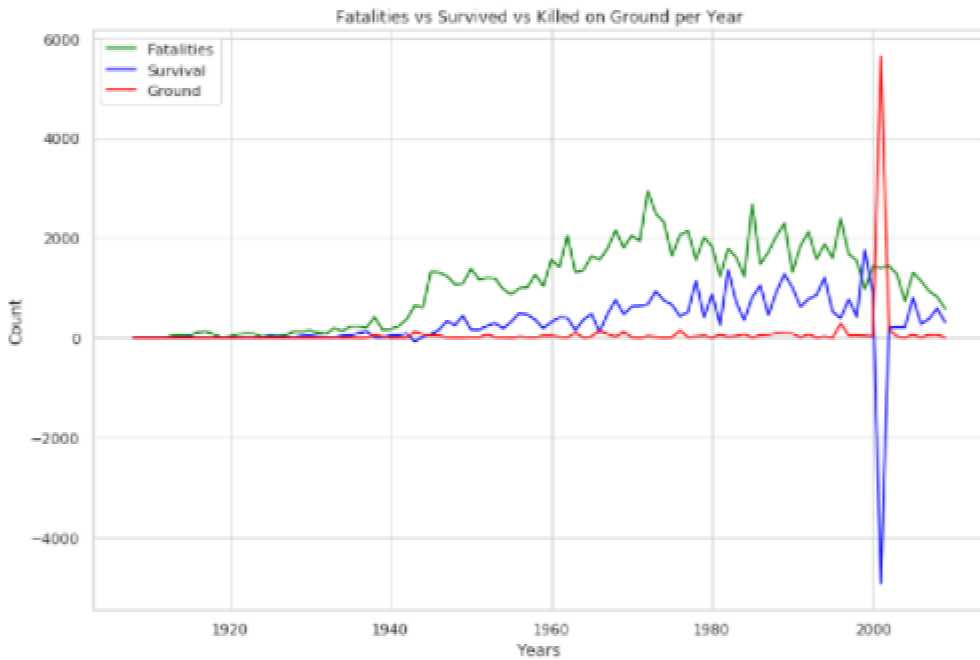
Fatalities vs Survived vs Killed on Ground

```
[ ] #summarise
    FSG_per_year = data[['Year', 'Fatalities', 'Survived', 'Ground']].groupby('Year').sum()
    FSG_per_year = FSG_per_year.reset_index()
```

```
[ ] #plot
    sns.lineplot(x = 'Year', y = 'Fatalities', data = FSG_per_year, color = 'green')
    sns.lineplot(x = 'Year', y = 'Survived', data = FSG_per_year, color = 'blue')
    sns.lineplot(x = 'Year', y = 'Ground', data = FSG_per_year, color = 'red')
    plt.legend(['Fatalities', 'Survival', 'Ground'])
    plt.xlabel('Years')
    plt.ylabel('Count')
    plt.title('Fatalities vs Survived vs Killed on Ground per Year')
```

```
Text(0.5,1,'Fatalities vs Survived vs Killed on Ground per Year')
```

```
Text(0.5,1,'Fatalities vs Survived vs Killed on Ground per Year')
```



```
[ ] oper_list = Counter(data['Operator']).most_common(10)
    operators = []
    crashes = []
    for tpl in oper_list:
        if 'Military' not in tpl[0]:
            operators.append(tpl[0])
            crashes.append(tpl[1])
    print('Top 10 the worst operators')
    pd.DataFrame({'Count of crashes' : crashes}, index=operators)
```

Top 10 the worst operators

	Count of crashes
AEROFLOT	179
MILITARY - U.S. AIR FORCE	176
AIR FRANCE	70
DEUTSCHE LUFTHANSA	65
AIR TAXI	48
CHINA NATIONAL AVIATION CORPORATION	44
UNITED AIR LINES	44
MILITARY - U.S. ARMY AIR FORCES	43
PAN AMERICAN WORLD AIRWAYS	41
MILITARY - U.S. NAVY	36

```
[ ] loc_list = Counter(data['Location'].dropna()).most_common(10)
    locs = []
    crashes = []
    for loc in loc_list:
        locs.append(loc[0])
        crashes.append(loc[1])
    print('Top 10 the most dangerous locations')
    pd.DataFrame({'Crashes in this location' : crashes}, index=locs)
```

Top 10 the most dangerous locations

	Crashes in this location

```
[ ] loc_list = Counter(data['Location'].dropna()).most_common(10)
    locs = []
    crashes = []
    for loc in loc_list:
        locs.append(loc[0])
        crashes.append(loc[1])
    print('Top 10 the most dangerous locations')
    pd.DataFrame({'Crashes in this location' : crashes}, index=locs)
```

Top 10 the most dangerous locations

	Crashes in this location
Sao Paulo, Brazil	15
Moscow, Russia	15
Rio de Janeiro, Brazil	14
Bogota, Colombia	13
Manila, Philippines	13
Anchorage, Alaska	13
New York, New York	12
Cairo, Egypt	12
Chicago, Illinois	11
Near Moscow, Russia	9

SCREENSHOTS OF CODE AND OUTPUTS FOR PROPOSED WORK

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.cluster import MiniBatchKMeans
from sklearn.metrics import adjusted_rand_score
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

```
text_data = data['Summary'].dropna()
text_data = pd.DataFrame(text_data)
# for reproducibility
random_state = 0
```

```
documents = list(text_data['Summary'])
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)
```

```
model.cluster_centers_
```

```
array([[3.32853808e-04, 5.61597890e-03, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.52139182e-04, 9.42550033e-03, 8.27526103e-05, ...,
        2.97713352e-04, 2.25400527e-04, 0.00000000e+00],
       [0.00000000e+00, 8.77272673e-03, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 1.34328619e-02, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 3.41261664e-03, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

```
# predict cluster labels for new dataset
model.predict(X)

# to get cluster labels for the dataset used while
# training the model (used for models that does not
# support prediction on new dataset).
model.labels_
```

```
array([1, 1, 1, ..., 1, 2, 0], dtype=int32)
```

```
print ('Most Common Terms per Cluster:')

order_centroids = model.cluster_centers_.argsort()[:,::-1] #sort cluster centers by proximity to centroid
terms = vectorizer.get_feature_names()

for i in range(5):
    print("\n")
    print('Cluster %d:' % i)
    for j in order_centroids[i, :10]: #replace 10 with n words per cluster
        print ('%s' % terms[j]),
    print
```

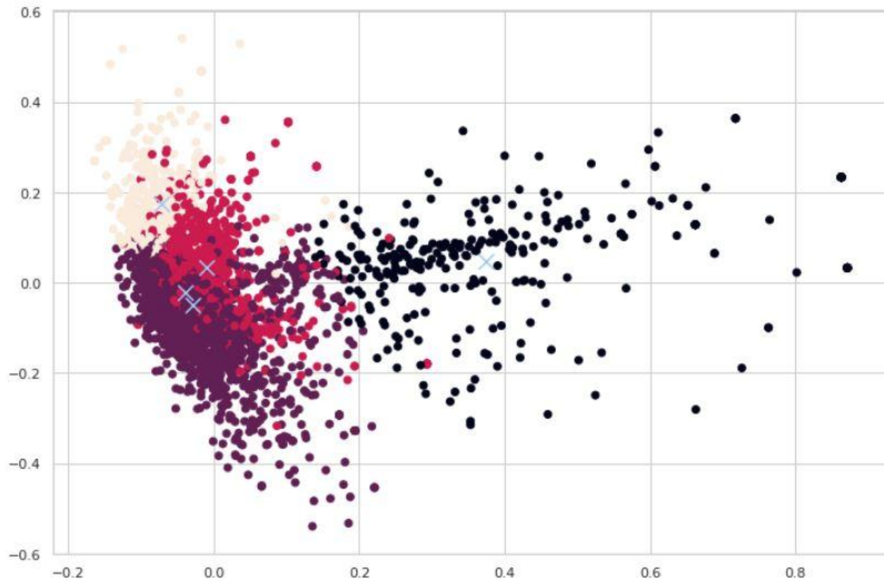
```
Most Common Terms per Cluster:


Cluster 0:
en
route
crashed
disappeared
mountain
plane
cargo
weather
flight
pilot


Cluster 1:
aircraft
approach
crashed
flight
pilot
weather
runway
mountain
conditions
struck
```

```
Cluster 2:
crashed
plane
taking
cargo
attempting
land
mountain
shortly
sea
fog

Cluster 3:
midair
collision
killed
aboard
dc
cessna
avoid
piper
air
mid

Cluster 4:
takeoff
engine
crashed
failure
shortly
plane
aircraft
lost
failed
runway
```

```
In [46]: # reduce the features to 2D
         pca = PCA(n_components=2, random_state=random_state)
         reduced_features = pca.fit_transform(X.toarray())

         # reduce the cluster centers to 2D
         reduced_cluster_centers = pca.transform(model.cluster_centers_)
```

```
In [47]: plt.scatter(reduced_features[:,0], reduced_features[:,1], c=model.predict(X))
         plt.scatter(reduced_cluster_centers[:, 0], reduced_cluster_centers[:,1], marker='x', s=150, c='b')
```

```
Out[47]: <matplotlib.collections.PathCollection at 0x7fbc81316128>
```




```
: print("\n")
  print("Prediction")

  Y = vectorizer.transform(["engine failure"])
  prediction = model.predict(Y)
  print(prediction)

  Y = vectorizer.transform(["terrorism"])
  prediction = model.predict(Y)
  print(prediction)
```

```
Prediction
[4]
[1]
```

```
]: import matplotlib.pyplot as plt

   from sklearn.cluster import DBSCAN
   from sklearn.preprocessing import StandardScaler
   from sklearn.preprocessing import normalize
   from sklearn.decomposition import PCA
```

```
]: text_data = data['Summary'].dropna()
   text_data = pd.DataFrame(text_data)
```

```
]: documents = list(text_data['Summary'])
   vectorizer = TfidfVectorizer(stop_words='english') # Stop words are like "a", "the", or "in" which don't have significant meaning
   X = vectorizer.fit_transform(documents)
```

```
]: # Scaling the data to bring all the attributes to a comparable level
   scaler = StandardScaler(with_mean=False)
   X_scaled = scaler.fit_transform(X)

   # Normalizing the data so that
   # the data approximately follows a Gaussian distribution
   X = normalize(X_scaled)
```

```
]: # for reproducibility
   random_state = 0
   model = DBSCAN(eps = 0.9, min_samples = 8)
   model.fit(X)
   labels = model.labels_
   print(labels)
```

```
[-1 -1 -1 ... -1 -1 -1]
```

```
In [54]: print(model.core_sample_indices_)
```

```
[  32   34   42   43   46   51   52   58   92  106  110  131  138  158
  160  174  189  214  220  226  234  235  236  242  243  258  260  274
  287  288  303  318  325  328  338  360  361  374  386  393  409  420
  423  425  427  428  432  435  450  457  469  474  484  495  504  505
  507  529  538  546  578  594  603  620  622  624  636  640  645  654
  659  661  680  690  692  705  719  735  752  757  759  786  794  800
  803  805  811  822  832  845  847  865  870  872  883  885  887  892
  895  903  905  907  912  919  938  948  973  991  997 1003 1019 1028
 1033 1036 1042 1051 1070 1086 1090 1113 1120 1140 1145 1170 1178 1192
 1229 1234 1237 1271 1295 1301 1337 1359 1363 1368 1371 1379 1380 1386
 1423 1444 1462 1473 1481 1496 1501 1525 1557 1584 1596 1606 1612 1615
 1659 1670 1673 1676 1682 1684 1688 1707 1716 1728 1734 1738 1758 1776
 1816 1832 1840 1843 1852 1858 1864 1866 1873 1883 1944 1946 1952 1967
 1969 1970 1977 1981 1988 1991 1998 2039 2044 2079 2081 2105 2125 2127
 2157 2203 2212 2215 2240 2241 2251 2261 2262 2268 2320 2337 2340 2342
 2345 2401 2437 2443 2458 2476 2480 2481 2486 2518 2519 2548 2552 2561
 2562 2565 2566 2569 2571 2574 2604 2627 2643 2645 2650 2695 2715 2721
 2729 2740 2749 2755 2758 2767 2808 2810 2824 2835 2838 2839 2855 2860
 2863 2877 2884 2891 2895 2902 2905 2913 2914 2917 2924 2946 2948 2959
 2973 2979 2985 3007 3029 3032 3058 3075 3089 3114 3142 3181 3185 3225
 3233 3261 3268 3270 3286 3306 3330 3342 3344 3358 3365 3379 3394 3401
 3403 3410 3414 3421 3450 3468 3474 3481 3491 3515 3520 3555 3566 3623
 3625 3673 3905 3919 3953 4038 4084 4097 4138 4151 4181 4247 4272 4317
 4397 4559 4786 4799]
```

```
print(model.components_)
```

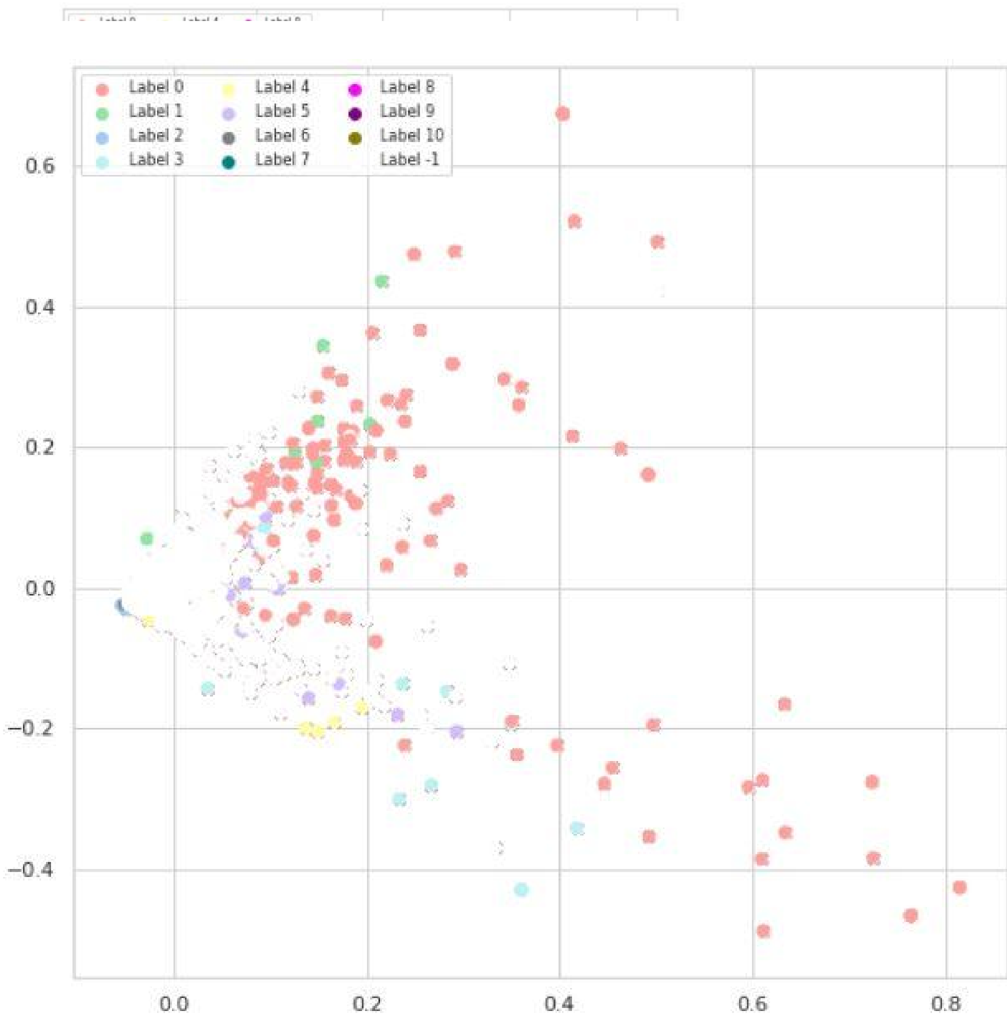
```
  (0, 9032)	0.6217119776519099
  (0, 6498)	0.7487436954292788
  (0, 2320)	0.22990671020885523
  (1, 7237)	0.6691737639603969
  (1, 3047)	0.6805373267354987
  (1, 2320)	0.2984550561588392
  (2, 3276)	0.7726360951652866
  (2, 3070)	0.6348491666905912
  (3, 3276)	0.7726360951652866
  (3, 3070)	0.6348491666905912
  (4, 3276)	0.7726360951652866
  (4, 3070)	0.6348491666905912
  (5, 3509)	0.9571571365227329
  (5, 2320)	0.2895690176859438
  (6, 3118)	0.8649452768425215
  (6, 6389)	0.5018661854197928
  (7, 3276)	0.7726360951652866
  (7, 3070)	0.6348491666905912
  (8, 5157)	0.6180416718107372
  (8, 7237)	0.1975390992198605
  (8, 3047)	0.20089360603919063
  (8, 6427)	0.1458245797315558
  (8, 7975)	0.7138750048258337
  (8, 2320)	0.08810348840083808
  (9, 4038)	0.7795121635936113
  :	:
  (319, 1006)	0.9436206566775635
  (319, 2320)	0.3310287846870783
  (320, 1646)	0.6322092754873192
  (320, 850)	0.549747906831004
  (320, 6427)	0.46730685040997816
  (320, 2320)	0.28233486940623337
  (321, 9032)	0.4537988333864844
  (321, 6498)	0.5465215850828127
  (321, 4828)	0.4882504565739478
  (321, 1006)	0.47836287520109655
  (321, 2320)	0.1678130720185628
  (322, 7237)	0.6691737639603969
```

```
(322, 3047)   0.6805373267354987
(322, 2320)   0.2984550561588392
(323, 7237)   0.6691737639603969
(323, 3047)   0.6805373267354987
(323, 2320)   0.2984550561588392
(324, 2712)   0.8667115652327125
(324, 7237)   0.3497297183095045
(324, 3047)   0.3556686475120007
(325, 1646)   0.5139815387861801
(325, 7237)   0.5146490080897417
(325, 3047)   0.5233885113780353
(325, 6427)   0.37991706761009325
(325, 2320)   0.22953619356273075
```

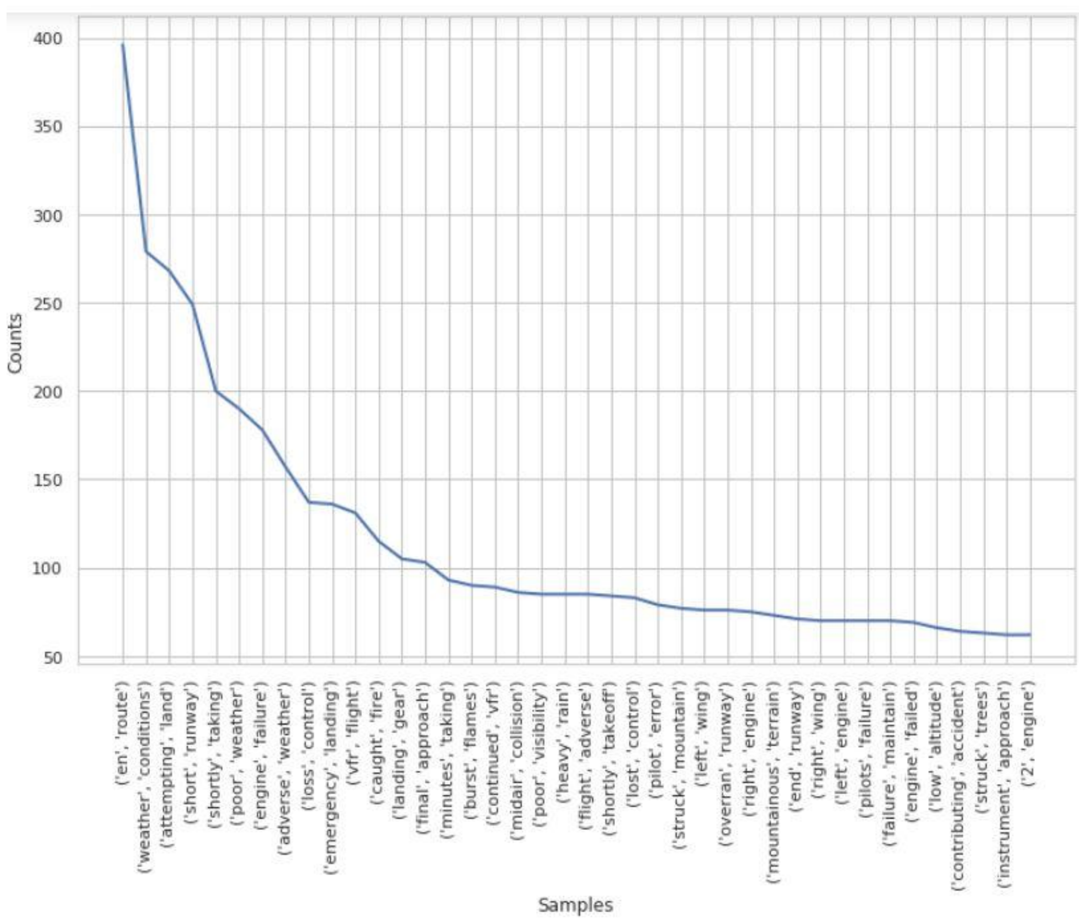
```
pca = PCA(n_components=2, random_state=random_state)
X_principal = pca.fit_transform(X.toarray())
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
print(X_principal.head())
```

```
         P1        P2
0 -0.040980 -0.016299
1 -0.038564 -0.014936
2 -0.041132 -0.016514
3 -0.039490 -0.015027
4 -0.041064 -0.015480
```

```
Purple = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color = colors[9])

Olive = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color = colors[10])

k = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color = colors[-1])

plt.scatter(X_principal['P1'], X_principal['P2'], c = cvec)
plt.legend((r, g, b, c, y, m,grey,teal ,Fuchsia,Purple,Olive,k),
        ('Label 0', 'Label 1', 'Label 2', 'Label 3', 'Label 4', 'Label 5', 'Label 6', 'Label 7', 'Label 8', 'Label 9', 'Label 10'
        loc ='upper left',
        ncol = 3,
        fontsize = 8)

plt.show()
```



```
from nltk import bigrams
from nltk import FreqDist
from nltk.corpus import stopwords
from gensim import corpora, models
import string

def remove_punctuation(s):
    exclude = set(string.punctuation)
    s = ''.join([i for i in s if i not in exclude])
    return s

stop = stopwords.words('english')
stop.append('plane')
stop.append('crashed')
stop.append('aircraft')

t = data[['Summary','Fatalities']].dropna()
book = t['Summary'].str.lower().apply(remove_punctuation).str.split().values.sum()
wrd = [w for w in book if w not in stop]


bigrams = list(bigrams(wrd))
fdistBigram = FreqDist(bigrams)
fdistBigram.plot(40)
```



```
]: summary = data['Summary'].tolist()
punctuation = ['.', ',', ':']
texts = []

for text in summary:
    cleaned_text = str(text).lower()
    for mark in punctuation:
        cleaned_text = cleaned_text.replace(mark, '')
    texts.append(cleaned_text.split())
```

```
]: dictionary = corpora.Dictionary(texts)
```

```
]: word_list = []
for key, value in dictionary.dfs.items():
    if value > 100:
        word_list.append(key)

dictionary.filter_tokens(word_list)
corpus = [dictionary.doc2bow(text) for text in texts]
```

```
]: np.random.seed(76543)
lda = models.LdaModel(corpus, num_topics=10, id2word=dictionary, passes=5)
```

```
]: topics = lda.show_topics(num_topics=10, num_words=15, formatted=False)
for topic in topics:
    num = int(topic[0]) + 1
    print('Cause %d:' % num, end=' ')
    print(', '.join([pair[0] for pair in topic[1]]))
```

```
Cause 1: descending, radar, clearance, mountainous, ridge, passenger, adequate, management, their, remote, cruise, atc, jungle,
meteorological, been
Cause 2: oil, subsequent, canyon, destroyed, included, km, coordination, lines, missing, pressure, number, flap, turned, doubl
e, maintenance
Cause 3: stall, end, nose, attitude, warning, 1, snow, ils, airspeed, overloaded, rudder, captain's, until, missed, follow
Cause 4: side, positioning, thrust, cessna, use, maintenance, went, carrying, tower, problems, seconds, rolled, reverse, their,
angle
Cause 5: disappeared, impacted, contact, ceiling, later, they, meteorological, destination, days, along, training, factor, 10,
experience, island
Cause 6: rebels, missile, ocean, controlled, lake, hillside, icing, km, experiencing, only, military, indicated, surface-to-ai
r, unita, excessive
Cause 7: jet, other, military, avoid, winds, just, maneuver, midair, burst, see, fighter, transport, may, houses, been
Cause 8: international, cockpit, weight, they, cabin, center, pitch, airplane's, forest, door, follow, three, passenger, passen
gers, icing
Cause 9: thunderstorm, thunderstorms, windshear, autopilot, mountainous, investigation, dive, flightcrew's, went, el, entered,
activity, strong, island, taxi
Cause 10: rotor, carrying, tail, overran, rest, came, separation, main, blade, gain, contamination, loaded, leading, fracture,
improperly
```

CONCLUSION:

From this project we analysed the trends and patterns in Airplane crashes and the causes that lead to a very huge loss.we used python programming to visualize the correlation between the labels.we used feature engineering and clustering algorithms for model predictionThe study suggests that machine learning techniques make it possible to predict the cause of airplane crashes. This could lead to significant savings for planners as well as those investigating air crashes.

REFERENCES:

- https://www.kaggle.com/saurograndi/airplane-crashes-since-1908
- https://www.ijrte.org/wp-content/uploads/papers/v8i6s/F10270386S20.pdf
- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html