Import libraries and datasets

In [ ]:
```
#Import libraries
import scipy.stats as stats
import numpy as np
from numpy import nan, isnan, mean, std, hstack, ravel
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.pylab as pl
import matplotlib.gridspec as gridspec
import matplotlib.cm as cm
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler, normalize
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering, DBSCAN, KMeans, M
iniBatchKMeans
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.manifold import TSNE
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis a
s LDA
import umap
import plotly.io as plt_io
import plotly.graph_objects as go

from wordcloud import WordCloud, STOPWORDS
from PIL import Image
import urllib
import requests

from datetime import date, timedelta, datetime

import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

In [ ]:
```
data = pd.read_csv('https://github.com/apham15/large_csv/raw/main/Airp
lane_Crashes_and_Fatalities_Since_1908.csv', na_values='nan')
```

In [ ]:
```
data.head(5)
```

Out[ ]:

	Date	Time	Location	Operator	Flight #	Route	Type	Registration	cn/I
0	09/17/1908	17:18	Fort Myer, Virginia	Military - U.S. Army	NaN	Demonstration	Wright Flyer III	NaN	
1	07/12/1912	06:30	AtlantiCity, New Jersey	Military - U.S. Navy	NaN	Test flight	Dirigible	NaN	Na
2	08/06/1913	NaN	Victoria, British Columbia, Canada	Private	-	NaN	Curtiss seaplane	NaN	Na
3	09/09/1913	18:30	Over the North Sea	Military - German Navy	NaN	NaN	Zeppelin L-1 (airship)	NaN	Na
4	10/17/1913	10:30	Near Johannisthal, Germany	Military - German Navy	NaN	NaN	Zeppelin L-2 (airship)	NaN	Na

In [ ]:
```
## Count of instances and features
rows, columns = data.shape
print(data.shape)
```

```
(5268, 13)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5268 entries, 0 to 5267
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Date          5268 non-null   object 
 1   Time          3049 non-null   object 
 2   Location      5248 non-null   object 
 3   Operator      5250 non-null   object 
 4   Flight #      1069 non-null   object 
 5   Route         3562 non-null   object 
 6   Type          5241 non-null   object 
 7   Registration  4933 non-null   object 
 8   cn/In         4040 non-null   object 
 9   Aboard        5246 non-null   float64
 10  Fatalities    5256 non-null   float64
 11  Ground        5246 non-null   float64
 12  Summary       4878 non-null   object 
dtypes: float64(3), object(10)
memory usage: 535.2+ KB
```

EDA

Data Cleaning

In [ ]:
```
#find any null value
data.isnull().sum()
```

Out[ ]:
```
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
dtype: int64
```

In [ ]:
```
#cleaning up
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') #is it 11:2
0 or 14:20 or smth else?

data.Operator = data.Operator.str.upper() #just to avoid duplicates li
ke 'British Airlines' and 'BRITISH Airlines'
```

In [ ]:
```
# Transforming Time column to datetime format and splitting into two s
eparate ones
time = pd.to_datetime(data['Time'], format='%H:%M')
data['hour'] = time.dt.hour
data['Year'] = data['Date'].apply(lambda x: int(str(x)[-4:]))
```

In [ ]:
```
#fill null values
data['Aboard'] = data['Aboard'].fillna(0)
data['Fatalities'] = data['Fatalities'].fillna(0)
data['Ground'] = data['Ground'].fillna(0)
```

In [ ]:
```
data['Fatalities_percentage'] = data['Fatalities'] / data['Aboard']
data['Time1'] = data['Date'] + ' ' + data['Time'] #joining two rows
def todate(x):
  return datetime.strptime(x, '%m/%d/%Y %H:%M')
data['Time1'] = data['Time1'].apply(todate) #convert to date type

print('Date ranges from ' + str(data.Time1.min()) + ' to ' + str(data.
Time1.max()))
```

```
Date ranges from 1908-09-17 17:18:00 to 2009-06-08 00:00:00
```

In [ ]:
```
#determine the survived groups
data['Survived'] = data['Aboard'] - data['Fatalities'] - data['Ground'
]
data['Has_Survivors'] = 1 #1 is survive
data.loc[data['Survived'] == 0, 'Has_Survivors'] = 0 #0 is no survive
```

In [ ]:
```
#recheck data
data.head()
```

Out[ ]:

	Date	Time	Location	Operator	Flight #	Route	Type	Registration	cn/I
0	09/17/1908	17:18	Fort Myer, Virginia	MILITARY - U.S. ARMY	NaN	Demonstration	Wright Flyer III	NaN	
1	07/12/1912	06:30	AtlantiCity, New Jersey	MILITARY - U.S. NAVY	NaN	Test flight	Dirigible	NaN	Na
2	08/06/1913	00:00	Victoria, British Columbia, Canada	PRIVATE	-	NaN	Curtiss seaplane	NaN	Na
3	09/09/1913	18:30	Over the North Sea	MILITARY - GERMAN NAVY	NaN	NaN	Zeppelin L-1 (airship)	NaN	Na
4	10/17/1913	10:30	Near Johannisthal, Germany	MILITARY - GERMAN NAVY	NaN	NaN	Zeppelin L-2 (airship)	NaN	Na

In [ ]:
```
#group the core numerical feature with airline operator
operator_fa = data[['Operator','Fatalities']].groupby('Operator').agg(
['sum','count'])
```

Understand the dataset

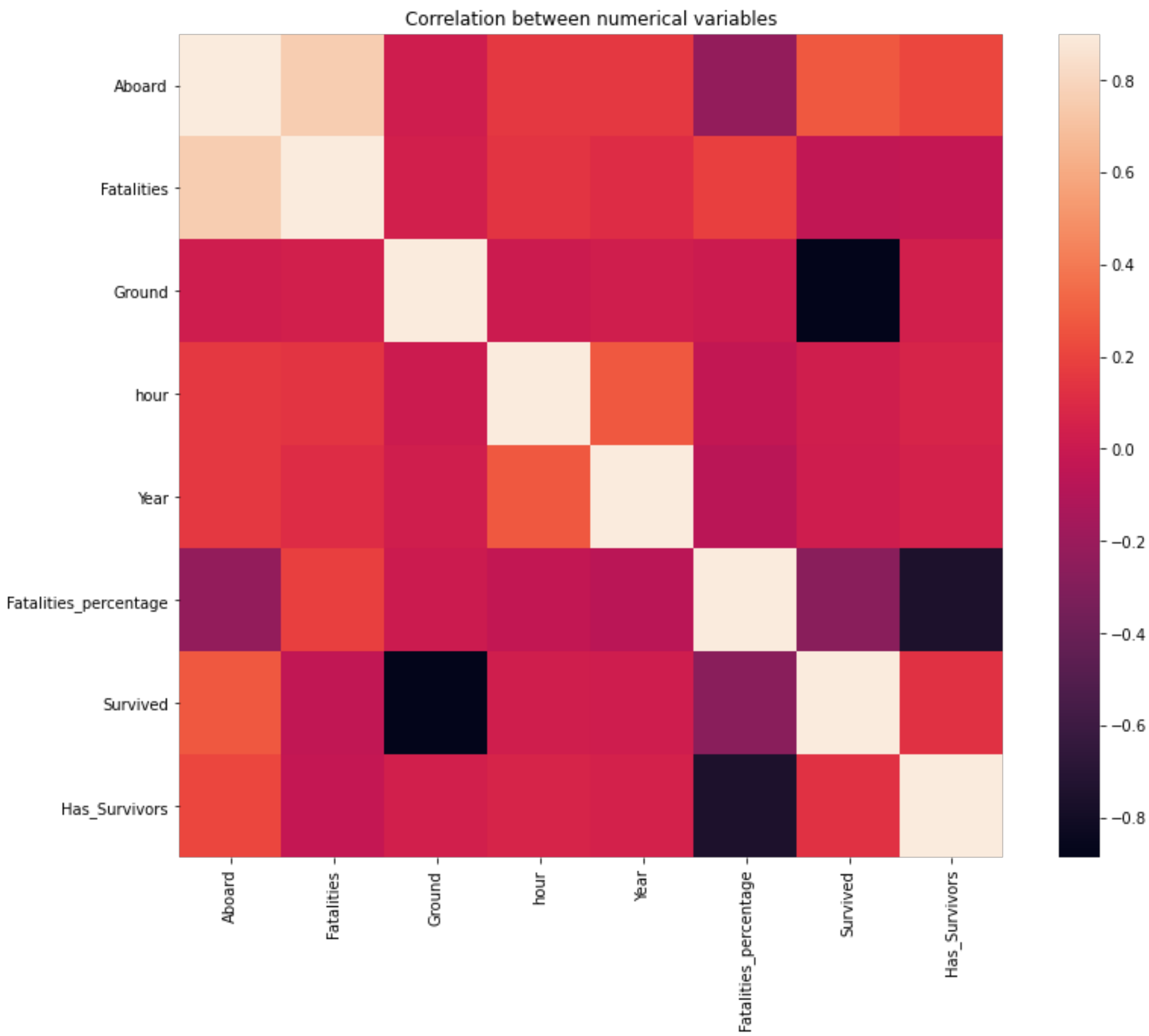
In [ ]:
```
# Univariate analysis
#statistical information for numerical data
data.describe()
```

Out[ ]:

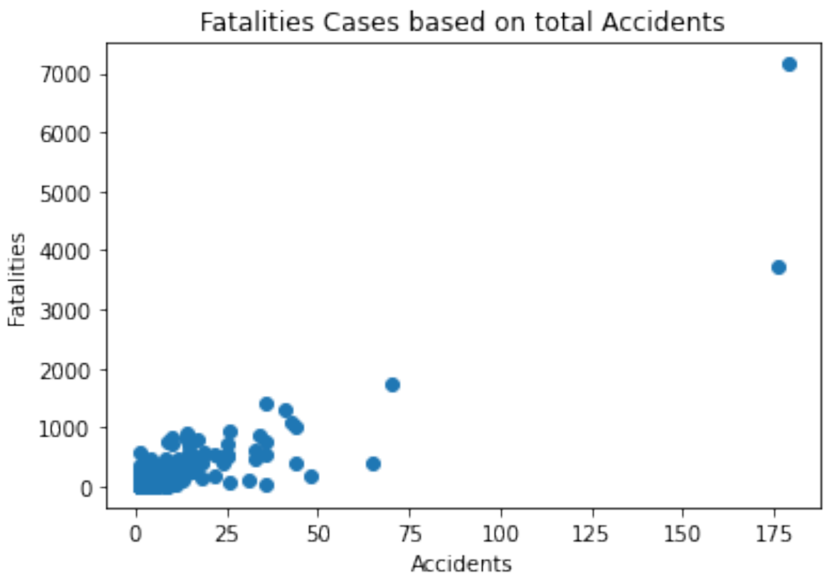
	Aboard	Fatalities	Ground	hour	Year	Fatalities_percentage
count	5268.000000	5268.000000	5268.000000	5268.000000	5268.000000	5254.000000
mean	27.439446	20.022589	1.602126	7.431283	1971.300304	inf
std	43.023370	33.175910	53.875057	7.827140	22.387541	NaN
min	0.000000	0.000000	0.000000	0.000000	1908.000000	0.000000
25%	5.000000	3.000000	0.000000	0.000000	1954.000000	0.805389
50%	13.000000	9.000000	0.000000	6.000000	1973.000000	1.000000
75%	30.000000	23.000000	0.000000	14.000000	1990.000000	1.000000
max	644.000000	583.000000	2750.000000	23.000000	2009.000000	inf

In [ ]:
```
#Correlation among variables
corr = data.corr()
plt.subplots(figsize=(13,10))
plt.title('Correlation between numerical variables')
sns.heatmap(corr, vmax=0.9, cmap="rocket", square=True)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3c7ac12208>

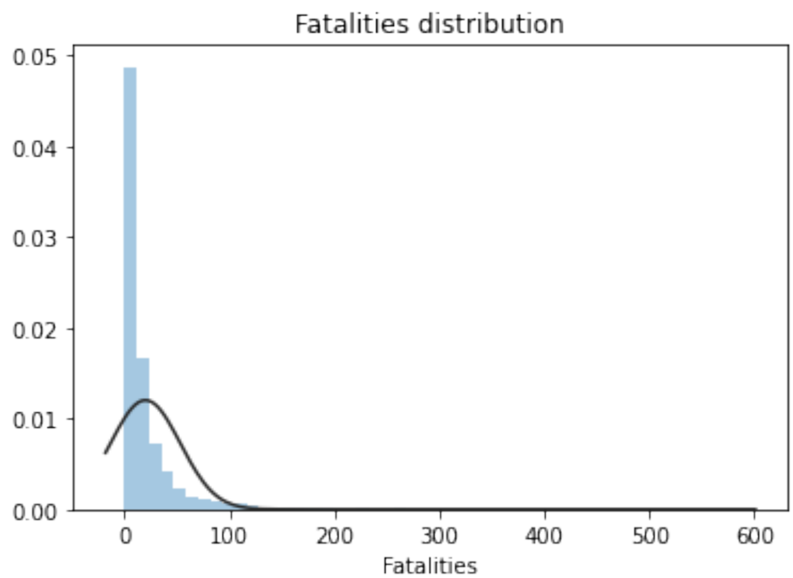


In [ ]:
```
#fatalities based on total accidents
X = operator_fa['Fatalities','count']
Y = operator_fa['Fatalities','sum']
plt.scatter(X, Y,label='Operators')
plt.title('Fatalities Cases based on total Accidents')
plt.ylabel('Fatalities')
plt.xlabel('Accidents');
```



In [ ]:
```
#Fatailities distribution
y = data['Fatalities']
plt.title('Fatalities distribution')
sns.distplot(y, kde=False, fit=stats.norm)
print("Skewness: %f" % data['Fatalities'].skew())
print("Kurtosis: %f" % data['Fatalities'].kurt())
```

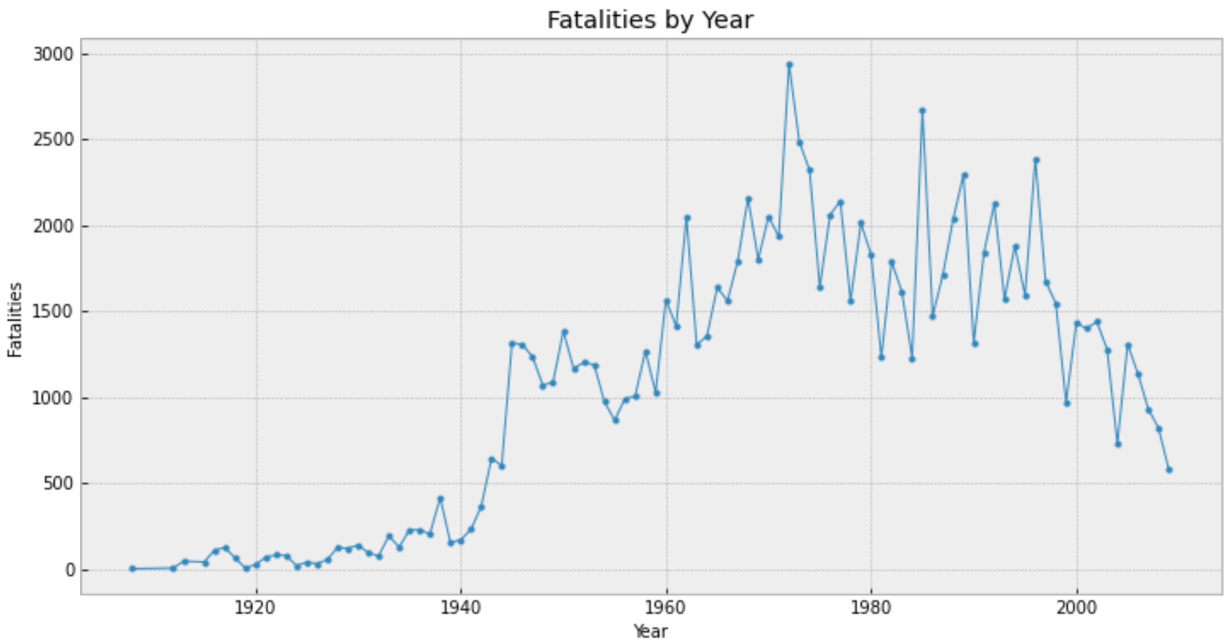
```
Skewness: 4.952818
Kurtosis: 42.889113
```



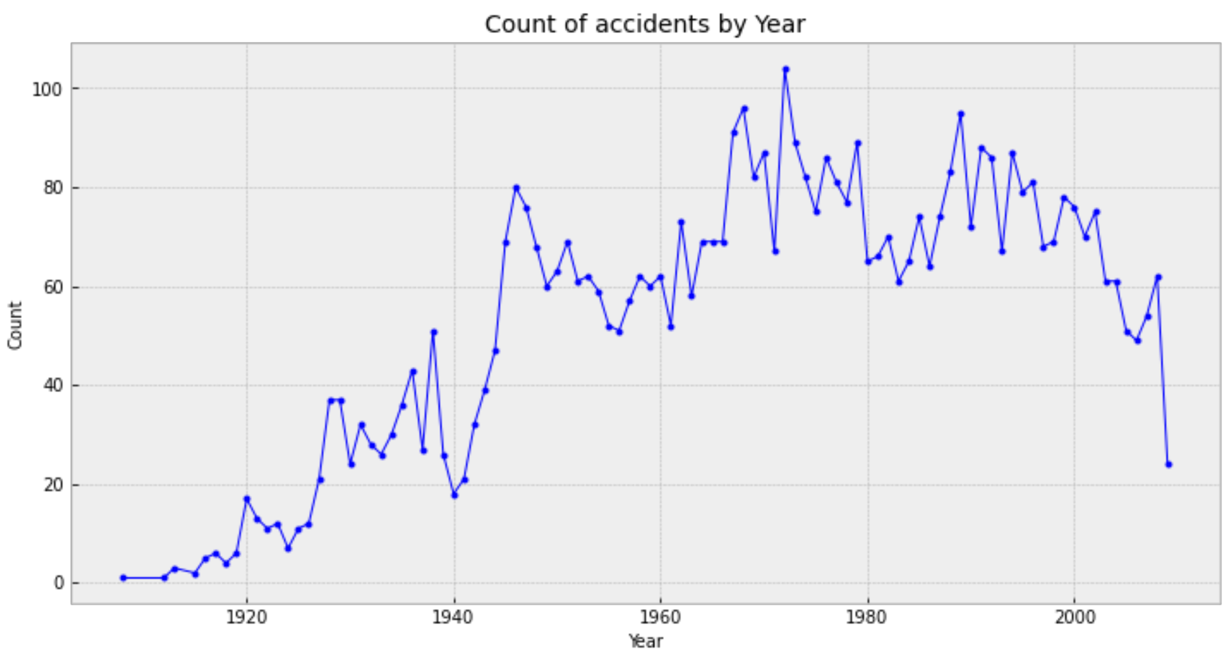
In [ ]:

```
#Line Plot with Fatalities per Year
yearly = data[['Year','Fatalities']].groupby('Year').agg(['sum','count'])
plt.style.use('bmh')
plt.figure(figsize=(12,6))
yearly['Fatalities','sum'].plot(title='Fatalities by Year',marker = ".", linewidth=1)
plt.xlabel('Year', fontsize=10)
plt.ylabel('Fatalities', fontsize=10)
```

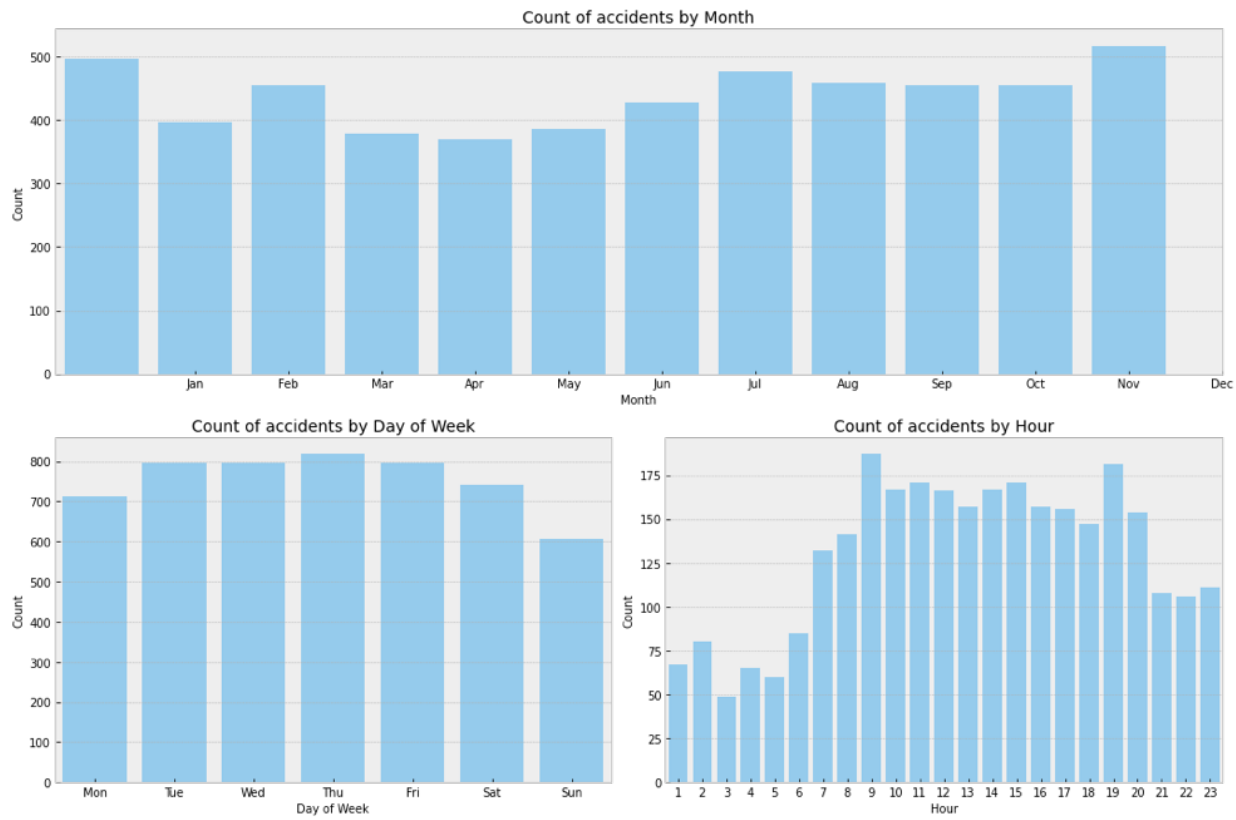
Out[ ]: Text(0, 0.5, 'Fatalities')



In [ ]:
```
#Line plot with Acident by year
Temp = data.groupby(data.Time1.dt.year)[['Date']].count() #Temp is goi
ng to be temporary data frame
Temp = Temp.rename(columns={"Date": "Count"})

plt.figure(figsize=(12,6))
plt.style.use('bmh')
plt.plot(Temp.index, 'Count', data=Temp, color='blue', marker = ".", l
inewidth=1)
plt.xlabel('Year', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.title('Count of accidents by Year', loc='Center', fontsize=14)
plt.show()
```

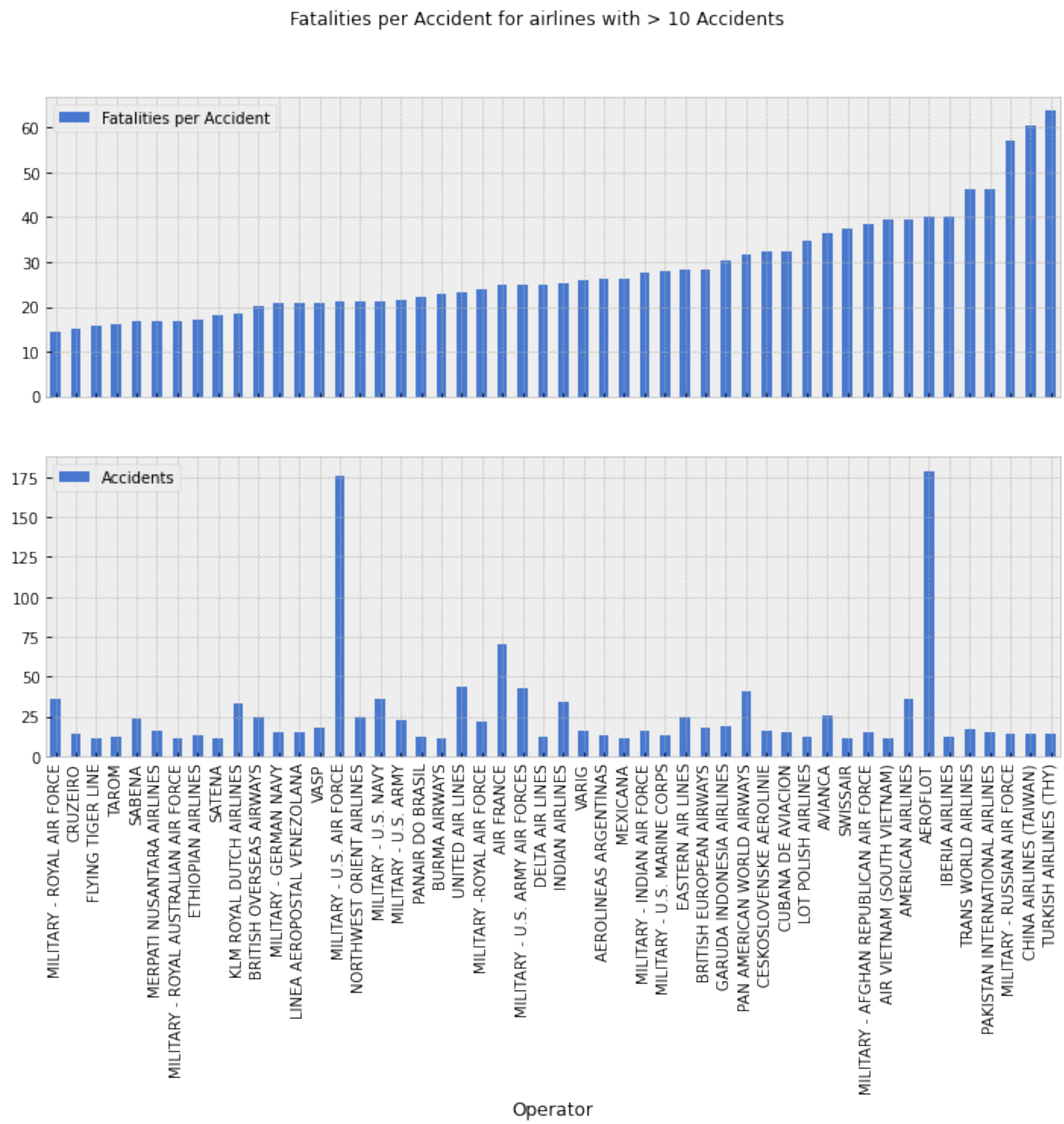


In [ ]:
```
#Bar Graph with No. of Accidents per Month, Day of Week and Hour
gs = gridspec.GridSpec(2, 2)
pl.figure(figsize=(15,10))
plt.style.use('seaborn-muted')
ax = pl.subplot(gs[0, :]) # row 0, col 0
sns.barplot(data.groupby(data.Time1.dt.month)[['Date']].count().index,
'Date', data=data.groupby(data.Time1.dt.month)[['Date']].count(), colo
r='lightskyblue', linewidth=2)
plt.xticks(data.groupby(data.Time1.dt.month)[['Date']].count().index,
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
'Nov', 'Dec'])
plt.xlabel('Month', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.title('Count of accidents by Month', loc='Center', fontsize=14)

ax = pl.subplot(gs[1, 0])
sns.barplot(data.groupby(data.Time1.dt.weekday)[['Date']].count().inde
x, 'Date', data=data.groupby(data.Time1.dt.weekday)[['Date']].count(),
color='lightskyblue', linewidth=2)
plt.xticks(data.groupby(data.Time1.dt.weekday)[['Date']].count().index
, ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'])
plt.xlabel('Day of Week', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.title('Count of accidents by Day of Week', loc='Center', fontsize=
14)

ax = pl.subplot(gs[1, 1])
sns.barplot(data[data.Time1.dt.hour != 0].groupby(data.Time1.dt.hour)[
['Date']].count().index, 'Date', data=data[data.Time1.dt.hour != 0].gr
oupby(data.Time1.dt.hour)[['Date']].count(),color ='lightskyblue', lin
ewidth=2)
plt.xlabel('Hour', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.title('Count of accidents by Hour', loc='Center', fontsize=14)
plt.tight_layout()
```

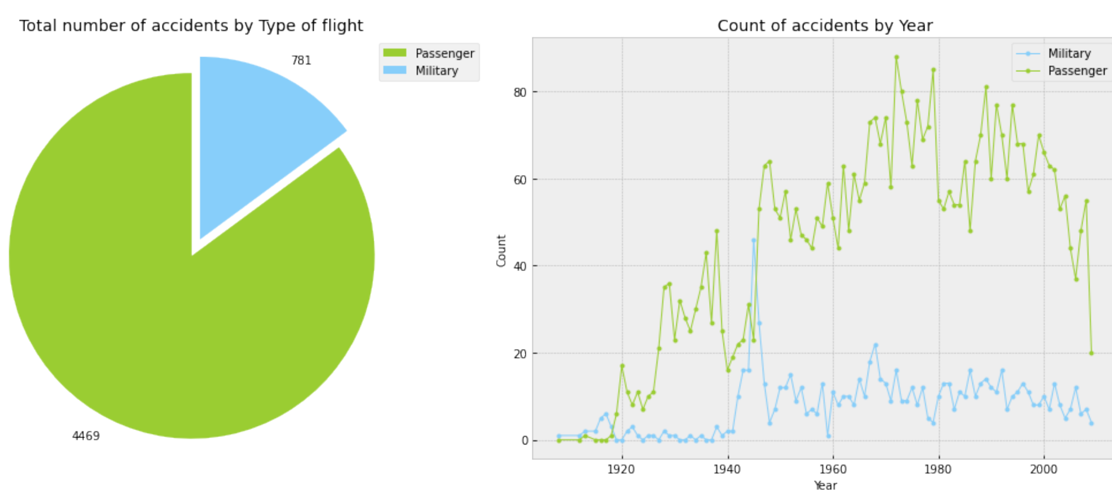

Count of accidents by Month
Count
0
100
200
300
400
500
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month
Count of accidents by Day of Week
Count
0
100
200
300
400
500
600
700
800
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Day of Week
Count of accidents by Hour
Count
0
25
50
75
100
125
150
175
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Hour

In [ ]:
```
matplotlib.rcParams['figure.figsize'] = (12.0, 8.0)
#Lets take a look at the proportion of fatalities per accident for spe
cific operators.
#This bears out some interesting statistics. Thanks for the suggestion
on kaggle and stackoverflow
props = operator_fa['Fatalities'].reset_index()
props['Fatalities per Accident'] = props['sum']/props['count']
props.columns = ['Operator','Fatalities','Accidents','Fatalities per A
ccident']

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)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3c610616d8>

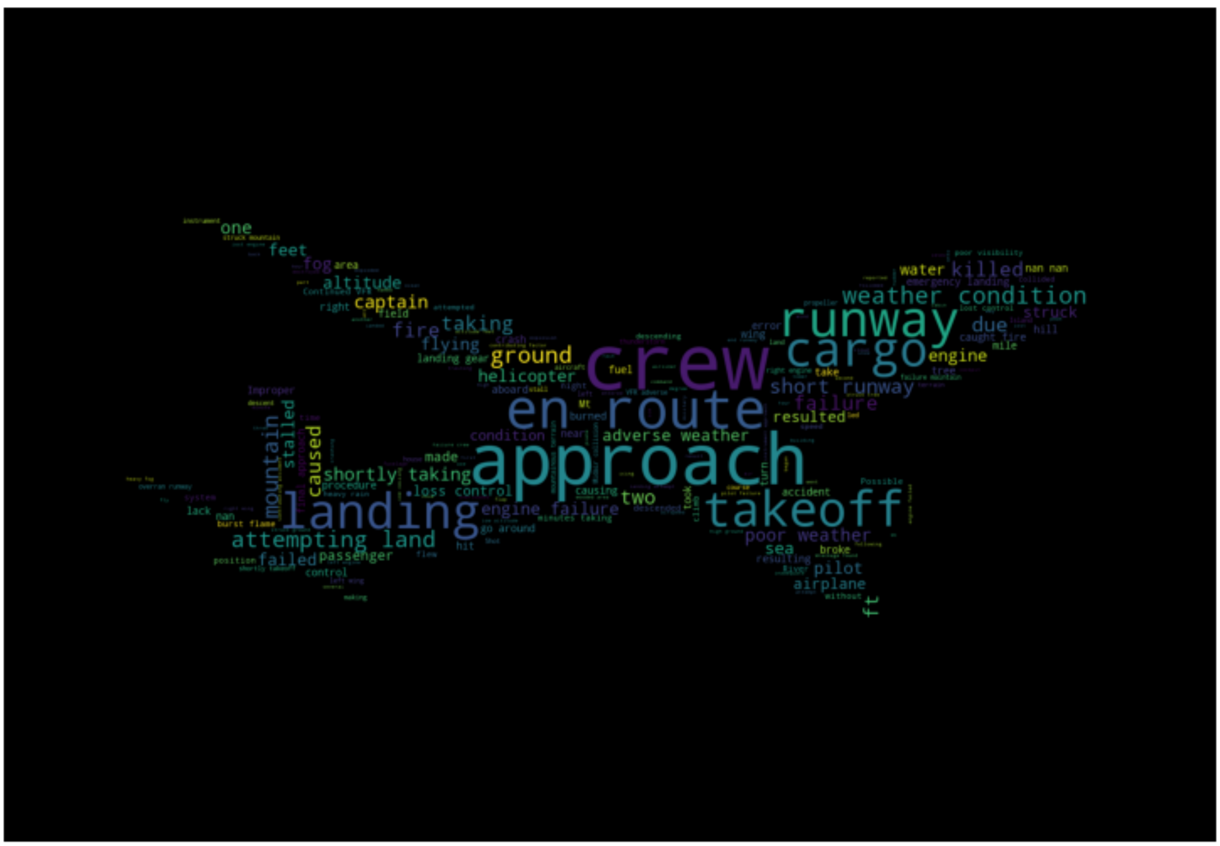


In [ ]:
```
Temp = data.copy()
Temp['isMilitary'] = Temp.Operator.str.contains('MILITARY')
Temp = Temp.groupby('isMilitary')[['isMilitary']].count()
Temp.index = ['Passenger', 'Military']

Temp2 = data.copy()
Temp2['Military'] = Temp2.Operator.str.contains('MILITARY')
Temp2['Passenger'] = Temp2.Military == False
Temp2 = Temp2.loc[:, ['Time1', 'Military', 'Passenger']]
Temp2 = Temp2.groupby(Temp2.Time1.dt.year)[['Military', 'Passenger']].aggregate(np.count_nonzero)

colors = ['yellowgreen', 'lightskyblue']
plt.figure(figsize=(15,6))
plt.subplot(1, 2, 1)
patches, texts = plt.pie(Temp.isMilitary, explode=[0,0.1], colors=colors, labels=Temp.isMilitary, startangle=90)
plt.legend(patches, Temp.index, loc="best", fontsize=10)
plt.axis('equal')
plt.title('Total number of accidents by Type of flight', loc='Center', fontsize=14)

plt.subplot(1, 2, 2)
plt.plot(Temp2.index, 'Military', data=Temp2, color='lightskyblue', marker = ".", linewidth=1)
plt.plot(Temp2.index, 'Passenger', data=Temp2, color='yellowgreen', marker = ".", linewidth=1)
plt.legend(fontsize=10)
plt.xlabel('Year', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.title('Count of accidents by Year', loc='Center', fontsize=14)
plt.tight_layout()
```

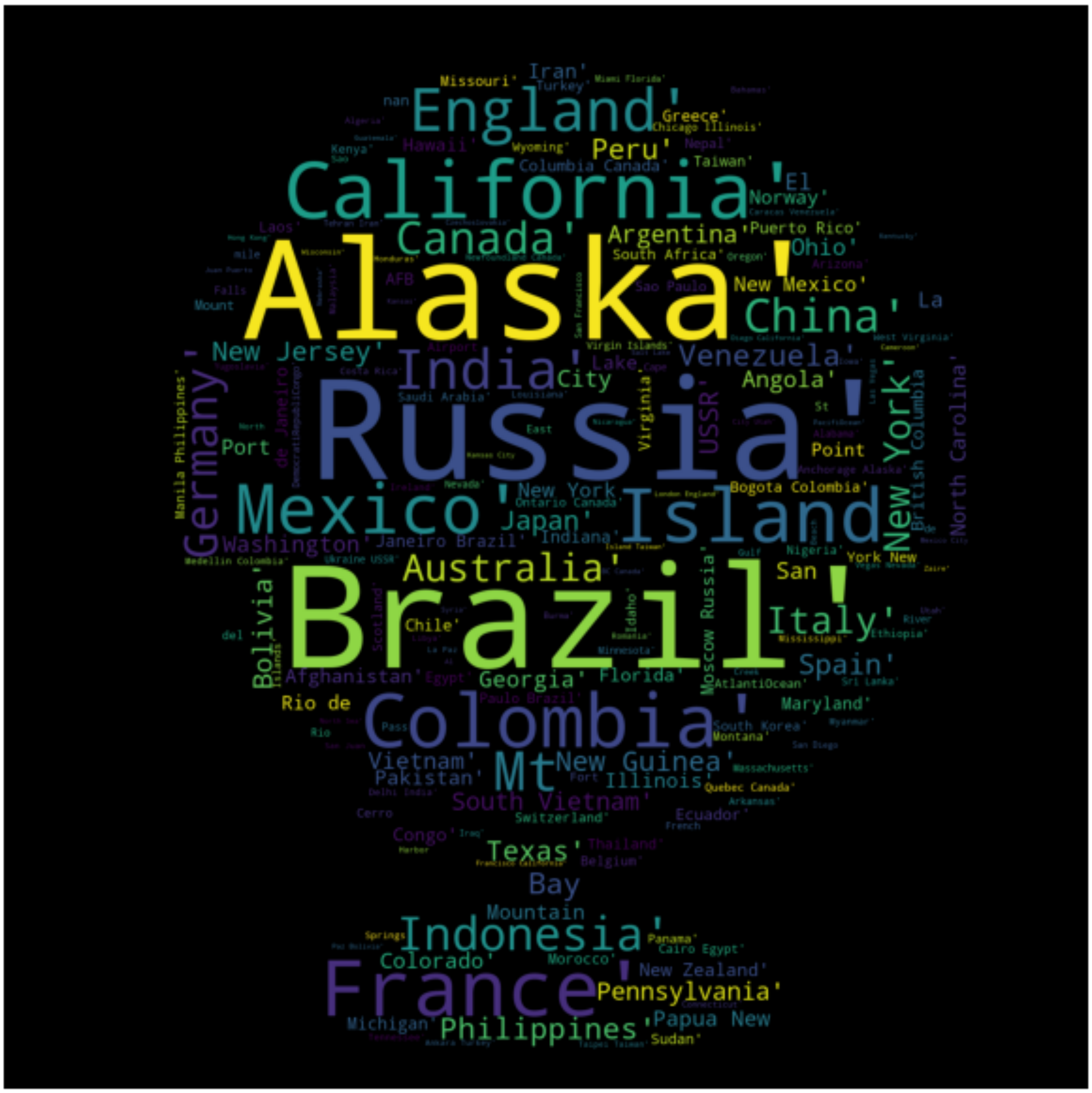


In [ ]:
```
text = str(data.Summary.tolist())
mask = np.array(Image.open(requests.get('https://i.pinimg.com/original
s/65/f5/b4/65f5b4c86574dd2090e9697398ef7a87.jpg', stream=True).raw))

stopwords = set(STOPWORDS)
newStopword = ['aircraft', 'pilot', 'en route', 'airport', 'flight', '
crashed', 'plane crashed', 'plane']
stopwords.update(newStopword)

# Write a function takes in text and mask and generates a wordcloud.
def generate_wordcloud(mask):
    word_cloud = WordCloud( background_color='black', stopwords=stopwo
rds, mask=mask)
    word_cloud.generate(text)
    plt.figure(figsize=(10,8),facecolor = 'white', edgecolor='blue')
    plt.imshow(word_cloud)
    plt.axis('off')
    plt.tight_layout(pad=0)
```

In [ ]:
```
#Run the following to generate wordcloud
generate_wordcloud(mask)
```



In [ ]:
```
text = str(data.Location.tolist())
mask = np.array(Image.open(requests.get('https://i5.walmartimages.com/
asr/c90e8d9d-8e28-4430-9930-e038c723b471.1910bb58304eede77fd067448bd91
498.jpeg', stream=True).raw))

stopwords = set(STOPWORDS)
newStopword = ['Near']
stopwords.update(newStopword)
#Run the following to generate wordcloud
generate_wordcloud(mask)
```



In [ ]:
```
#Splitting out the country from the location to see if we can find som
e interesting statistics about where the most crashes have taken place
.
fatalities = operator_fa['Fatalities','sum'].sort_values(ascending=Fal
se)
totalfatal = fatalities.sum()
fatalprop = fatalities/totalfatal

s = data['Location'].str[0:].str.split(',', expand=True)
data['Country'] = s[3].fillna(s[2]).fillna(s[1]).str.strip()
```

In [ ]:
```
#put all the US's states into US columne so it's easier to assign them
a country
usNames = ['Virginia','New Jersey','Ohio','Pennsylvania', 'Maryland',
'Indiana', 'Iowa',
          'Illinois','Wyoming', 'Minnisota', 'Wisconsin', 'Nevada', 'N
Y','California',
          'WY','New York','Oregon', 'Idaho', 'Connecticut','Nebraska',
'Minnesota', 'Kansas',
          'Texas', 'Tennessee', 'West Virginia', 'New Mexico', 'Washin
gton', 'Massachusetts',
          'Utah', 'Ilinois','Florida', 'Michigan', 'Arkansas','Colorad
o', 'Georgia''Missouri',
          'Montana', 'Mississippi','Alaska','Jersey', 'Cailifornia', '
Oklahoma','North Carolina',
          'Kentucky','Delaware','D.C.','Arazona','Arizona','South Deko
ta','New Hampshire','Hawaii',
          'Washingon','Massachusett','Washington DC','Tennesee','Delew
are','Louisiana',
          'Massachutes', 'Louisana', 'New York (Idlewild)','Oklohoma',
'North Dakota','Rhode Island',
          'Maine','Alakska','Wisconson','Calilfornia','Virginia','Virg
inia.','CA','Vermont',
          'HI','AK','IN','GA','Coloado','Airzona','Alabama','Alaksa'
          ]
```

In [ ]:
```
#define and clean countries' names
#Decided to try and cleanse the country names.
afNames = ['Afghanstan'] #Afghanistan
anNames = ['off Angola'] #Angola
ausNames = ['Qld. Australia','Queensland  Australia','Tasmania','off A
ustralia'] #Australia
argNames = ['Aregntina'] #Argentina
azNames = ['Azores (Portugal)'] #Azores
baNames = ['Baangladesh'] #Bangladesh
bahNames = ['Great Inagua'] #Bahamas
berNames = ['off Bermuda'] #Bermuda
bolNames = ['Boliva','BO'] #Bolivia
```

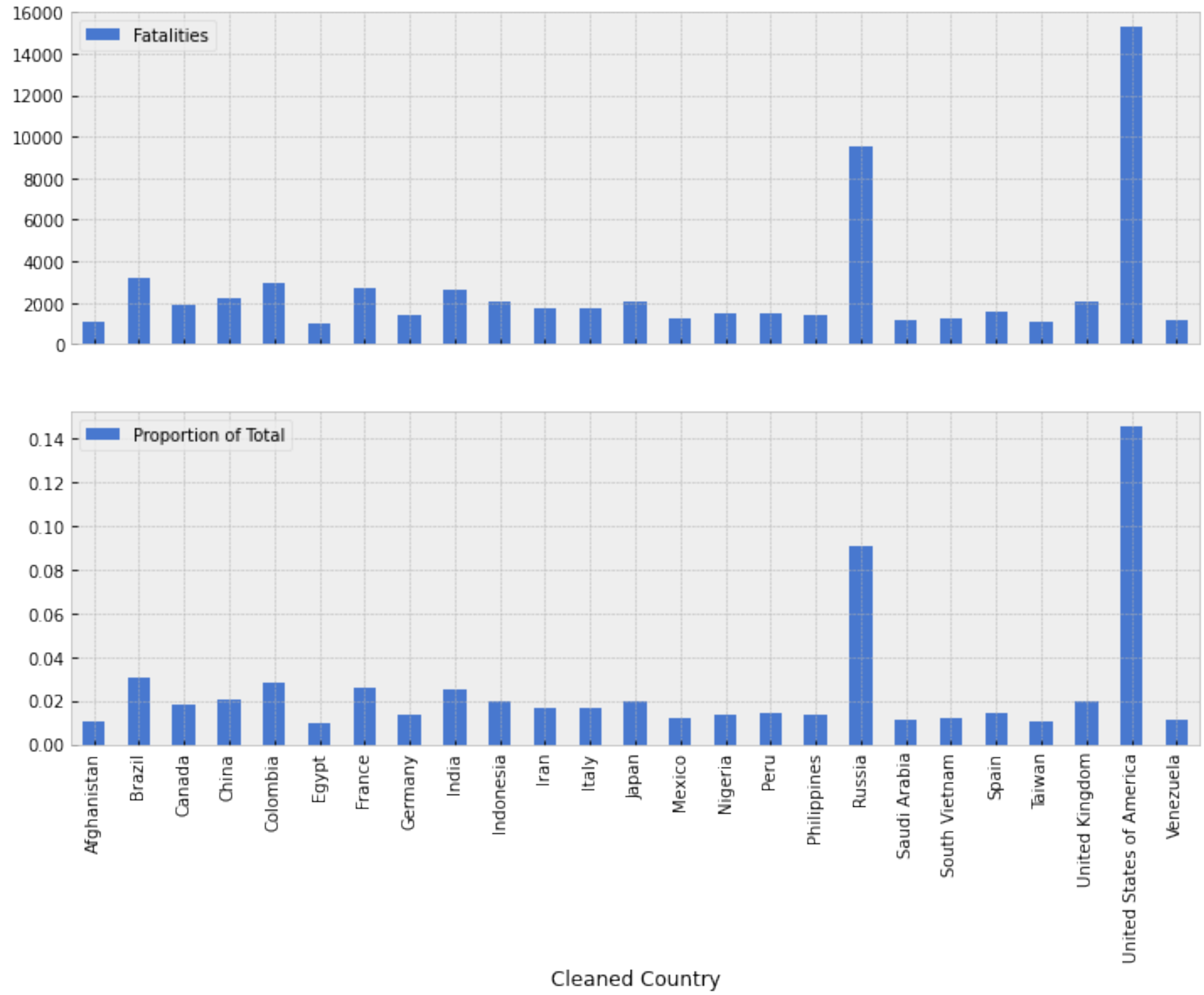
```
bhNames = ['Bosnia-Herzegovina'] #Bosnia Herzegovina
bulNames = ['Bugaria','Bulgeria'] #Bulgaria
canNames = ['British Columbia', 'British Columbia Canada','Canada2',
            'Saskatchewan','Yukon Territory'] #Canada
camNames = ['Cameroons','French Cameroons'] #Cameroon
caNames = ['Cape Verde Islands'] #Cape Verde
chNames = ['Chili'] #Chile
coNames = ['Comoro Islands', 'Comoros Islands'] #Comoros
djNames = ['Djbouti','Republiof Djibouti'] #Djibouti
domNames = ['Domincan Republic', 'Dominica'] #Dominican Republic
drcNames = ['Belgian Congo','Belgian Congo (Zaire)','Belgium Congo'
           'DR Congo','DemocratiRepubliCogo','DemocratiRepubliCongo',
            'DemocratiRepubliof Congo','DemoctratiRepubliCongo','Zaire
',
           'Zaïre'] #Democratic Republic of Congo
faNames = ['French Equitorial Africa'] #French Equatorial Africa
gerNames = ['East Germany','West Germany'] #Germany
grNames = ['Crete'] #Greece
haNames = ['Hati'] #Haiti
hunNames = ['Hunary'] #Hungary
inNames = ['Indian'] #India
indNames = ['Inodnesia','Netherlands Indies'] #Indonesia
jamNames = ['Jamacia'] #Jamaica
malNames = ['Malaya'] #Malaysia
manNames = ['Manmar'] #Myanmar
marNames = ['Mauretania'] #Mauritania
morNames = ['Morrocco','Morroco'] #Morocco
nedNames = ['Amsterdam','The Netherlands'] #Netherlands
niNames = ['Niger'] #Nigeria
philNames = ['Philipines','Philippine Sea', 'Phillipines',
            'off the Philippine island of Elalat'] #Philippines
romNames = ['Romainia'] #Romania
rusNames = ['Russian','Soviet Union','USSR'] #Russia
saNames = ['Saint Lucia Island'] #Saint Lucia
samNames = ['Western Samoa'] #Samoa
siNames = ['Sierre Leone'] #Sierra Leone
soNames = ['South Africa (Namibia)'] #South Africa
surNames = ['Suriname'] #Surinam
uaeNames = ['United Arab Emirates'] #UAE
ukNames = ['England', 'UK','Wales','110 miles West of Ireland'] #Unite
d Kingdom
uvNames = ['US Virgin Islands','Virgin Islands'] #U.S. Virgin Islands
wkNames = ['325 miles east of Wake Island']#Wake Island
yuNames = ['Yugosalvia'] #Yugoslavia
zimNames = ['Rhodesia', 'Rhodesia (Zimbabwe)'] #Zimbabwe

clnames = []
for country in data['Country'].values:
  if country in afNames:
      clnames.append('Afghanistan')
```

```
    elif country in anNames:
        clnames.append('Angola')
    elif country in ausNames:
        clnames.append('Australia')
    elif country in argNames:
        clnames.append('Argentina')
    elif country in azNames:
        clnames.append('Azores')
    elif country in baNames:
        clnames.append('Bangladesh')
    elif country in bahNames:
        clnames.append('Bahamas')
    elif country in berNames:
        clnames.append('Bermuda')
    elif country in bolNames:
        clnames.append('Bolivia')
    elif country in bhNames:
        clnames.append('Bosnia Herzegovina')
    elif country in bulNames:
        clnames.append('Bulgaria')
    elif country in canNames:
        clnames.append('Canada')
    elif country in camNames:
        clnames.append('Cameroon')
    elif country in caNames:
        clnames.append('Cape Verde')
    elif country in chNames:
        clnames.append('Chile')
    elif country in coNames:
        clnames.append('Comoros')
    elif country in djNames:
        clnames.append('Djibouti')
    elif country in domNames:
        clnames.append('Dominican Republic')
    elif country in drcNames:
        clnames.append('Democratic Republic of Congo')
    elif country in faNames:
        clnames.append('French Equatorial Africa')
    elif country in gerNames:
        clnames.append('Germany')
    elif country in grNames:
        clnames.append('Greece')
    elif country in haNames:
        clnames.append('Haiti')
    elif country in hunNames:
        clnames.append('Hungary')
    elif country in inNames:
        clnames.append('India')
    elif country in jamNames:
        clnames.append('Jamaica')
```

```
    elif country in malNames:
        clnames.append('Malaysia')
    elif country in manNames:
        clnames.append('Myanmar')
    elif country in marNames:
        clnames.append('Mauritania')
    elif country in morNames:
        clnames.append('Morocco')
    elif country in nedNames:
        clnames.append('Netherlands')
    elif country in niNames:
        clnames.append('Nigeria')
    elif country in philNames:
        clnames.append('Philippines')
    elif country in romNames:
        clnames.append('Romania')
    elif country in rusNames:
        clnames.append('Russia')
    elif country in saNames:
        clnames.append('Saint Lucia')
    elif country in samNames:
        clnames.append('Samoa')
    elif country in siNames:
        clnames.append('Sierra Leone')
    elif country in soNames:
        clnames.append('South Africa')
    elif country in surNames:
        clnames.append('Surinam')
    elif country in uaeNames:
        clnames.append('UAE')
    elif country in ukNames:
        clnames.append('United Kingdom')
    elif country in usNames:
        clnames.append('United States of America')
    elif country in uvNames:
        clnames.append('U.S. Virgin Islands')
    elif country in wkNames:
        clnames.append('Wake Island')
    elif country in yuNames:
        clnames.append('Yugoslavia')
    elif country in zimNames:
        clnames.append('Zimbabwe')
    else:
        clnames.append(country)
```

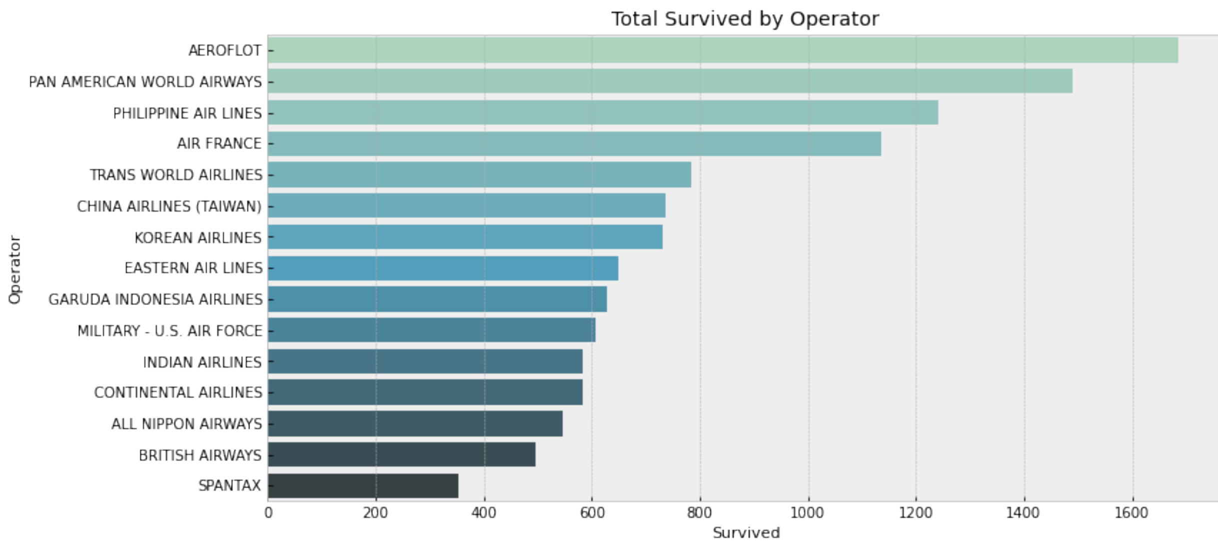
In [ ]:
```
#visuallize the highest fatalities by countries
data['Cleaned Country'] = clnames
fatalcountries = data[['Fatalities','Cleaned Country']].groupby(['Clea
ned Country']).agg('sum')
fatalcountries.reset_index(inplace = True)
fatalcountries['Proportion of Total'] = fatalcountries['Fatalities']/t
otalfatal

fig_c, (ax1,ax2) = plt.subplots(2,1,sharex = True)
fatalcountries[fatalcountries['Fatalities']>1000].plot(x = 'Cleaned Co
untry'
                                                    , y = 'Fatalities
'
                                                    , ax = ax1
                                                    , kind = 'bar'
                                                    , grid = True)
fatalcountries[fatalcountries['Fatalities']>1000].plot(x = 'Cleaned Co
untry'
                                                    , y = 'Proportion
of Total'
                                                    , ax = ax2
                                                    , kind = 'bar'
                                                    , grid = True)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3c5d7fb780>

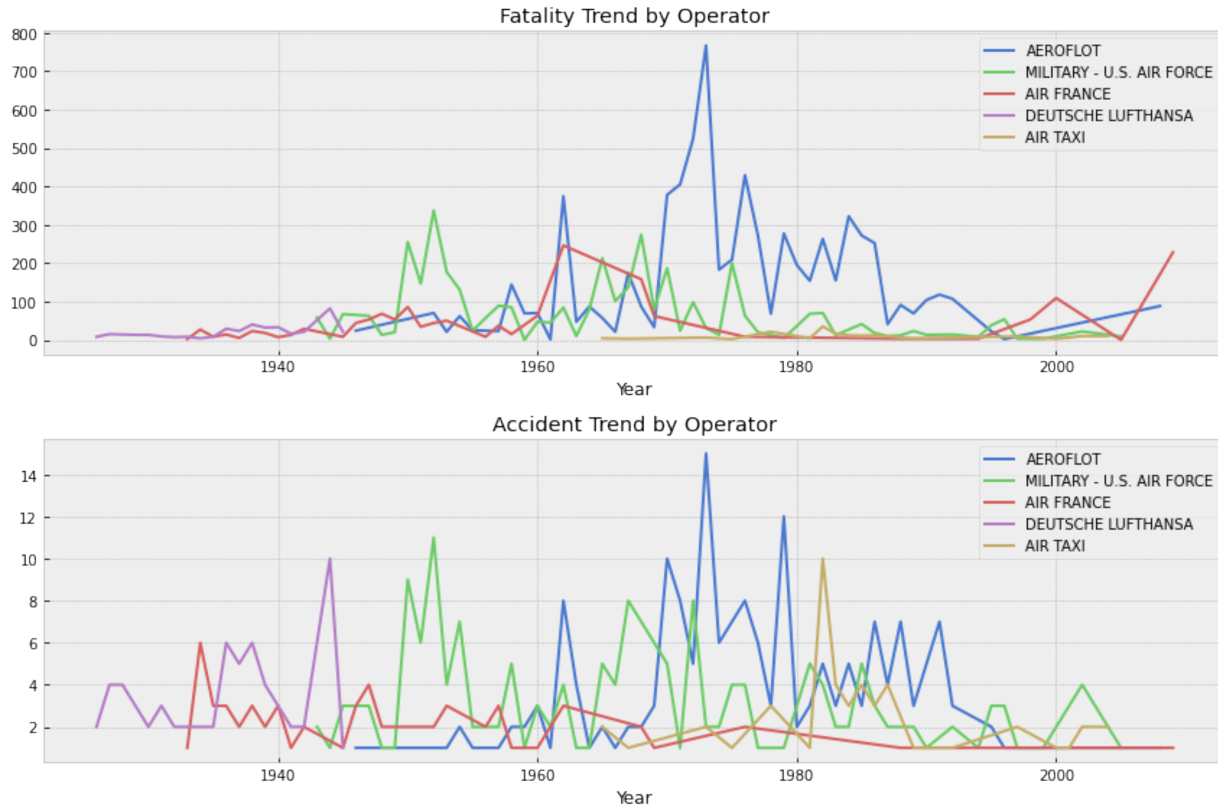


In [ ]:
```
data_t = data.groupby('Operator')[['Survived']].sum()
data_t = data_t.rename(columns={"Operator": "Survived"})
data_t = data_t.sort_values(by='Survived', ascending=False)
Prop_by_OpTOP = data_t.head(15)

plt.figure(figsize=(12,6))
sns.barplot(y=Prop_by_OpTOP.index, x="Survived", data=Prop_by_OpTOP, p
alette="GnBu_d", orient='h')
plt.xlabel('Survived', fontsize=11)
plt.ylabel('Operator', fontsize=11)
plt.title('Total Survived by Operator', loc='Center', fontsize=14)
```

Out[ ]:
```
Text(0.5, 1.0, 'Total Survived by Operator')
```



In [ ]:
```
accidents = operator_fa['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()
```



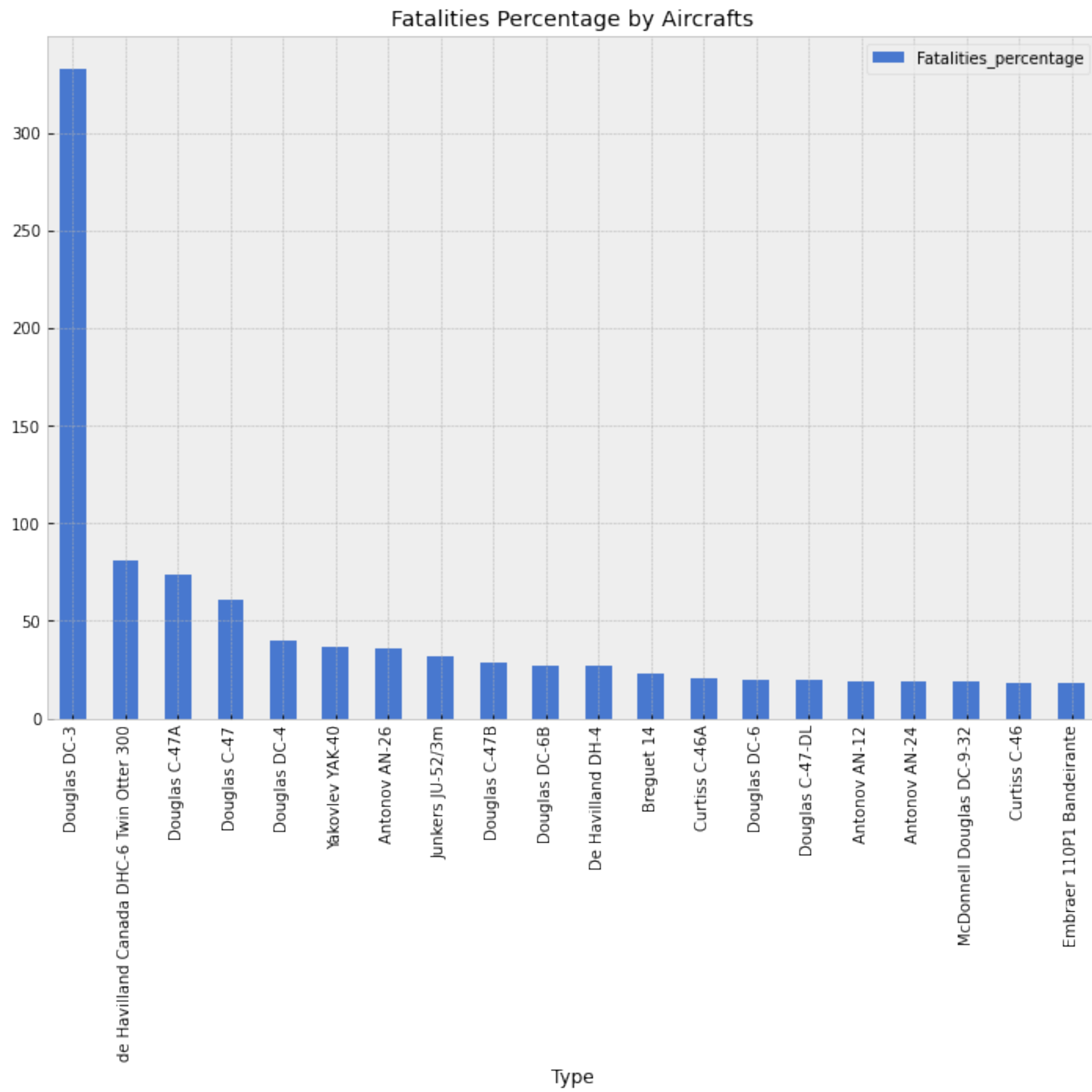
In [ ]:
```
fatalcountries.sort_values(by='Proportion of Total', ascending=False).
head(10)
```

Out[ ]:

	Cleaned Country	Fatalities	Proportion of Total
228	United States of America	15288.0	0.145153
181	Russia	9538.0	0.090560
30	Brazil	3205.0	0.030430
46	Colombia	2935.0	0.027867
72	France	2735.0	0.025968
98	India	2628.0	0.024952
45	China	2189.0	0.020784
227	United Kingdom	2071.0	0.019663
100	Indonesia	2050.0	0.019464
109	Japan	2050.0	0.019464

In [ ]:
```
data.groupby(['Type']).agg({'Fatalities_percentage':'count'}).sort_val
ues(by='Fatalities_percentage',ascending=False).head(20).plot.bar()
plt.title('Fatalities Percentage by Aircrafts', loc='Center', fontsize
=14)
```

Out[ ]:
```
Text(0.5, 1.0, 'Fatalities Percentage by Aircrafts')
```


Fatalities Percentage by Aircrafts
Fatalities_percentage
0
50
100
150
200
250
300
Douglas DC-3
de Havilland Canada DHC-6 Twin Otter 300
Douglas C-47A
Douglas C-47
Douglas DC-4
Yakovlev YAK-40
Antonov AN-26
Junkers JU-52/3m
Douglas C-47B
Douglas DC-6B
De Havilland DH-4
Breguet 14
Curtiss C-46A
Douglas DC-6
Douglas C-47-DL
Antonov AN-12
Antonov AN-24
McDonnell Douglas DC-9-32
Curtiss C-46
Embraer 110P1 Bandeirante
Type

In [ ]:
```
#route that has high fatalities rate
route= data.groupby('Route').agg({'Fatalities':['sum',lambda x:x.sum()
/ data['Fatalities'].sum()]})
route.columns=route.columns.map(''.join)
route.reset_index(inplace=True)
route.rename(columns={'Fatalitiessum':'Total Fatalities','Fatalities<l
ambda_0>':'% of Total Fatalities'}, inplace=True)

route.sort_values(by='Total Fatalities', ascending=False).head(7)
```

Out[ ]:

	Route	Total Fatalities	% of Total Fatalities
2968	Tenerife - Las Palmas / Tenerife - Las Palmas	583.0	0.005527
3012	Tokyo - Osaka	557.0	0.005281
3029	Training	457.0	0.004333
2276	Paris - London	375.0	0.003555
2057	New Delhi - Dhahran / Chimkent - New Delhi	349.0	0.003309
1957	Montreal - London	329.0	0.003119
2522	Riyadh - Jeddah	301.0	0.002854

In [ ]:

```
#Location that has most plane crashed
from collections import Counter
loc_list = Counter(data['Location'].dropna()).most_common(10)
locs = []
crashes = []
for loc in loc_list:
    locs.append(loc[0])
    crashes.append(loc[1])
pd.DataFrame({'Crashes in this location' : crashes}, index=locs)
```

Out[ ]:

	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

In [ ]:
```
# Creating a fun dataset based on Chinese Zodiac
# Return a bunch of tuples with the Zodiac and its Start/End Dates
def chinese_zodaics():
    start_date = pd.to_datetime("2/2/1908")
    end_date = pd.to_datetime("7/1/2009")
    animals = ['Monkey', 'Rooster', 'Dog', 'Pig', 'Rat', 'Ox', 'Tiger'
, 'Rabbit', 'Dragon', 'Snake', 'Horse', 'Goat']
    zodiacs = []
    while start_date < end_date:
        for a in animals:
            year_start = start_date
            year_end = year_start + pd.DateOffset(days=365)
            z = (a, start_date, year_end)
            zodiacs.append(z)
            start_date = year_end
    return zodiacs

zodiacs = chinese_zodaics()

# Apply the zodiacs to the accident dates
def match_zodiac(date):
    for z in zodiacs:
        animal, start, end, = z[0], z[1], z[2]
        if start <= date <= end:
            return animal
data_c=data
data_c.Date = pd.to_datetime(data_c.Date)
data_c['Zodiac'] = data_c.Date.apply(match_zodiac)
data_c['Year'] = pd.DatetimeIndex(data_c['Date']).year
data_c = data_c[['Zodiac', 'Year', 'Fatalities', 'Aboard','Ground']].d
ropna()
data_c = data_c[data.Fatalities > 1]
data_c
```

Out[ ]:

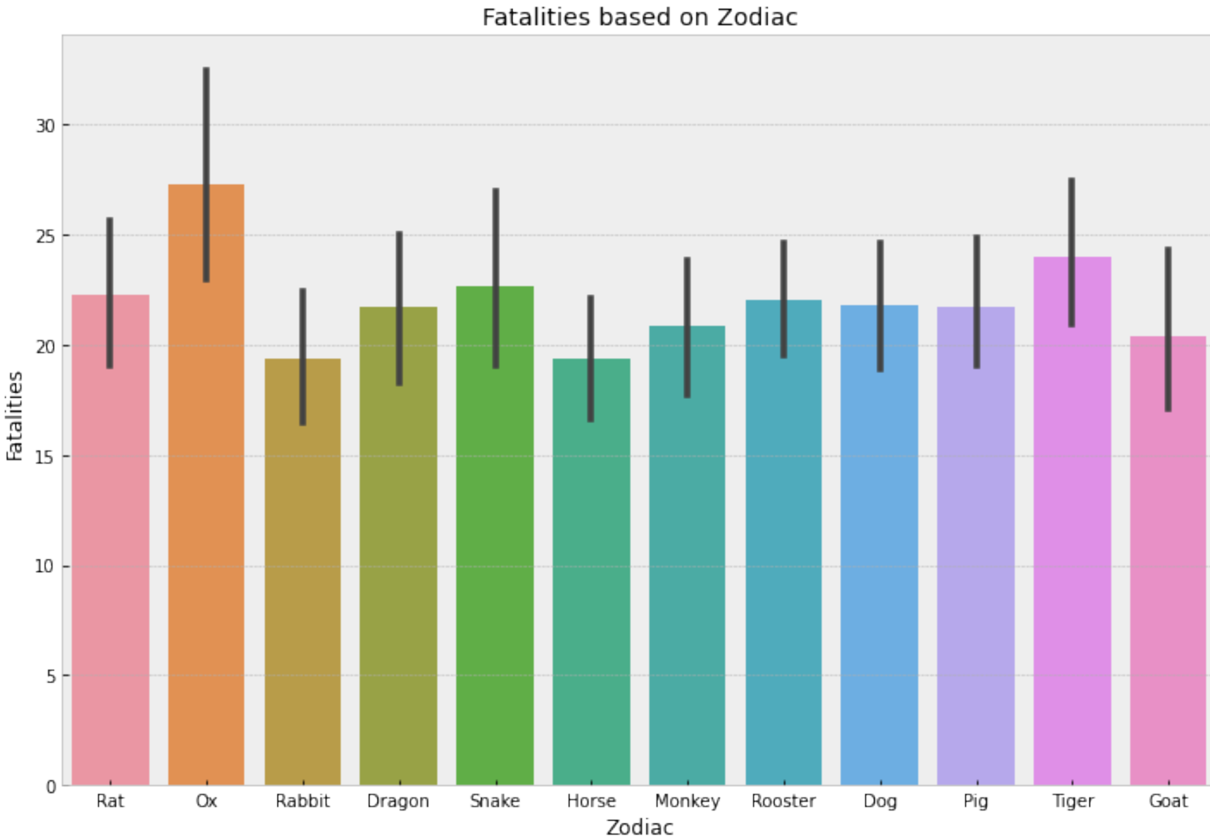
	Zodiac	Year	Fatalities	Aboard	Ground
1	Rat	1912	5.0	5.0	0.0
3	Ox	1913	14.0	20.0	0.0
4	Ox	1913	30.0	30.0	0.0
5	Rabbit	1915	21.0	41.0	0.0
6	Rabbit	1915	19.0	19.0	0.0
...	...	...	...	...	...
5262	Ox	2009	18.0	18.0	0.0
5263	Ox	2009	98.0	112.0	2.0
5264	Ox	2009	4.0	4.0	0.0
5265	Ox	2009	228.0	228.0	0.0
5267	Ox	2009	13.0	13.0	0.0

4789 rows × 5 columns

In [ ]:

```
sns.barplot (x='Zodiac', y='Fatalities', data = data_c)
plt.title('Fatalities based on Zodiac', loc='Center', fontsize=14)
```

Out[ ]: Text(0.5, 1.0, 'Fatalities based on Zodiac')



In [ ]:

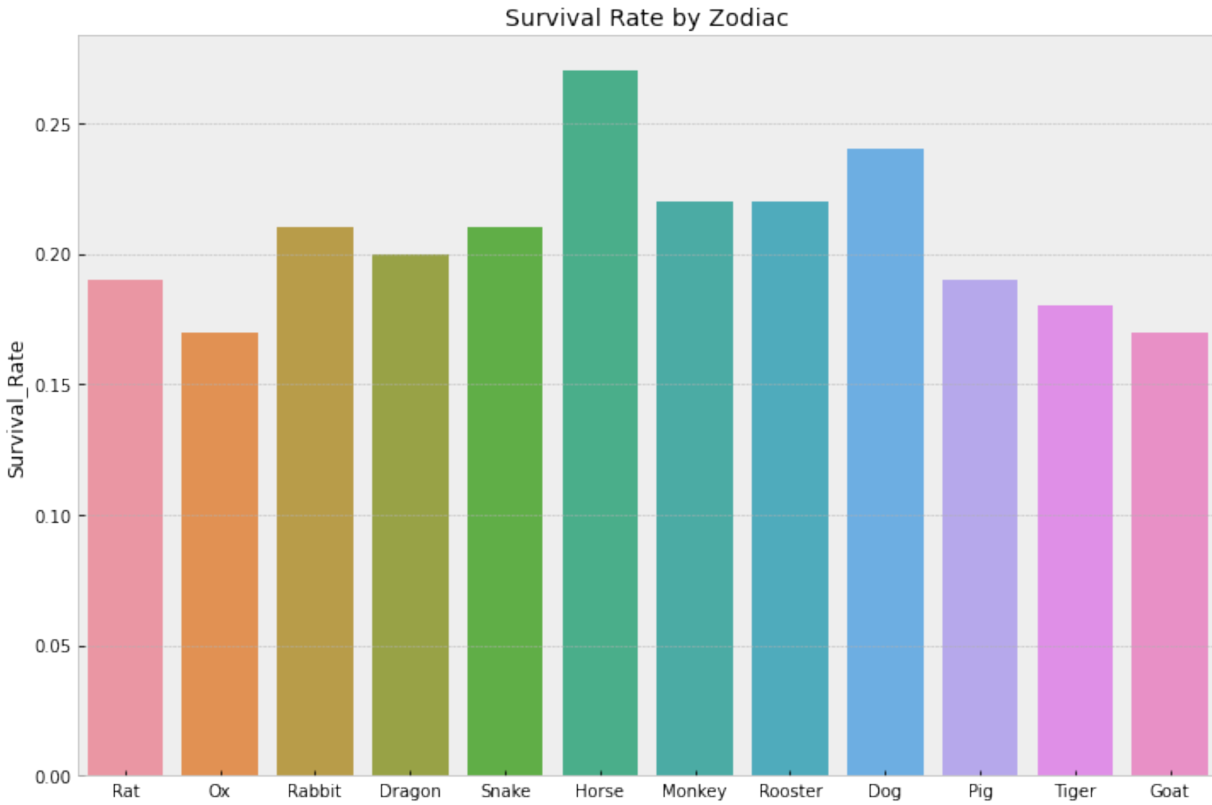
```
# Put key stats into a DataFrame
def zodiac_data(data):
    idx=[ 'Total_Accidents', 'Total_Deaths', 'Mean_Deaths', 'Death_Rat
e', 'Survival_Rate', 'Deadliest_Accident', 'Total_Survive']
    df = pd.DataFrame()
    for z in data.Zodiac.unique():
        zodiac = data[data.Zodiac == z]
        f = zodiac.Fatalities.dropna()
        a = zodiac.Aboard.dropna()
        g = zodiac.Ground.dropna()
        total_accidents = f.count()
        total_deaths = f.sum()
        mean_deaths = f.mean()
        death_rate = total_deaths / a.sum()
        survival_rate = 1 - death_rate
        deadliest = f.max()
        survive = sum(a-f-g)
        df[z] = [total_accidents, total_deaths, mean_deaths, death_rat
e, survival_rate, deadliest, survive]
    df.index = idx
    df = df.round(2).T
    return df

zodiac_comparison = zodiac_data(data_c)
zodiac_comparison
```

Out[ ]:

	Total_Accidents	Total_Deaths	Mean_Deaths	Death_Rate	Survival_Rate	Deadliest_Ac
Rat	448.0	9981.0	22.28	0.81	0.19	
Ox	372.0	10134.0	27.24	0.83	0.17	
Rabbit	359.0	6956.0	19.38	0.79	0.21	
Dragon	390.0	8476.0	21.73	0.80	0.20	
Snake	408.0	9241.0	22.65	0.79	0.21	
Horse	373.0	7209.0	19.33	0.73	0.27	
Monkey	405.0	8446.0	20.85	0.78	0.22	
Rooster	395.0	8692.0	22.01	0.78	0.22	
Dog	438.0	9533.0	21.76	0.76	0.24	
Pig	397.0	8628.0	21.73	0.81	0.19	
Tiger	390.0	9349.0	23.97	0.82	0.18	
Goat	414.0	8425.0	20.35	0.83	0.17	

In [ ]:
```
sns.barplot(x=zodiac_comparison.index, y='Survival_Rate', data=zodiac_
comparison)
plt.title('Survival Rate by Zodiac', loc='Center', fontsize=14)
```

Out[ ]:
```
Text(0.5, 1.0, 'Survival Rate by Zodiac')
```



Clustering : Fatalities Clustering

Data Preparation

In [ ]:
```
data_1 = data.drop(['Date','Time','Location','Summary', 'Fatalities_pe
rcentage','hour', 'Year', 'Time1'], axis = 1)
data_1.Country = pd.get_dummies(data_1.Country)
X = data_1.iloc[:,6:12]
X.isnull().any()
```

Out[ ]:
```
Aboard           False
Fatalities       False
Ground           False
Survived         False
Has_Survivors    False
Country          False
dtype: bool
```

In [ ]:
```
cols = ['Aboard', 'Fatalities', 'Ground','Survived','Country']
for col in cols:
        X[col] = X[col].to_numpy()
        X[col] = X[col].astype('int64')
```

In [ ]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5268 entries, 0 to 5267
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Aboard         5268 non-null   int64
 1   Fatalities     5268 non-null   int64
 2   Ground         5268 non-null   int64
 3   Survived       5268 non-null   int64
 4   Has_Survivors  5268 non-null   int64
 5   Country        5268 non-null   int64
dtypes: int64(6)
memory usage: 247.1 KB
```

Feature Engineering

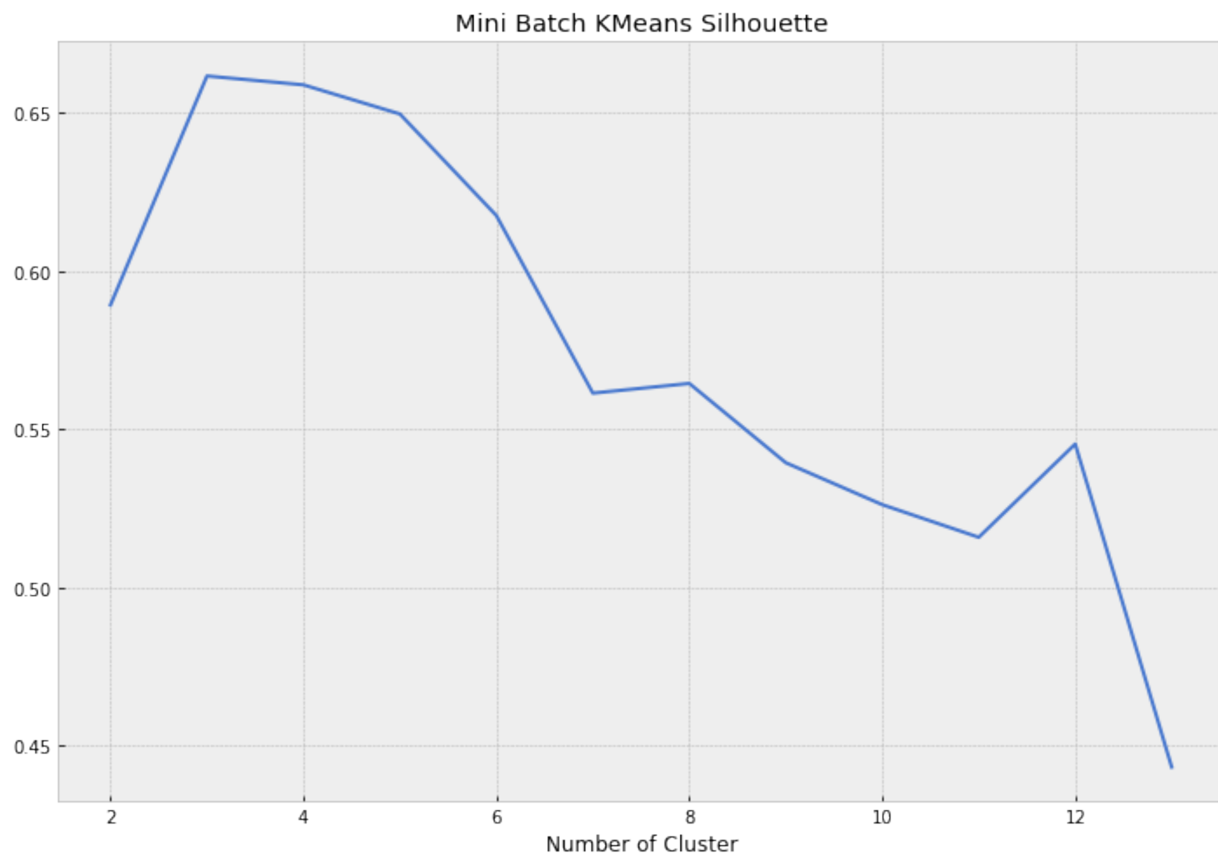
In [ ]:
```
# Standarizing the features
scaler = StandardScaler()
X_stan = scaler.fit_transform(X)
```

Clustering

K-Means

In [ ]:
```
#Apply KMean with the evaluation based on Silhouette Score
%%time
sil_k = np.arange(12,dtype="double")
for k in np.arange(12):
  minikm = MiniBatchKMeans(n_clusters=k+2, random_state=0)
  minikm.fit(X_stan)
  sil_k[k] = metrics.silhouette_score(X_stan,minikm.labels_,metric='eu
clidean')

print(sil_k)

plt.title("Mini Batch KMeans Silhouette")
plt.xlabel("Number of Cluster")
plt.plot(np.arange(2,14,1),sil_k)
```

```
[0.58926807 0.66160985 0.65880994 0.64960927 0.61751482 0.56141789
 0.56447616 0.53943237 0.52618258 0.51587685 0.54533361 0.44328984]
CPU times: user 13.8 s, sys: 40.1 s, total: 53.8 s
Wall time: 4.72 s
```

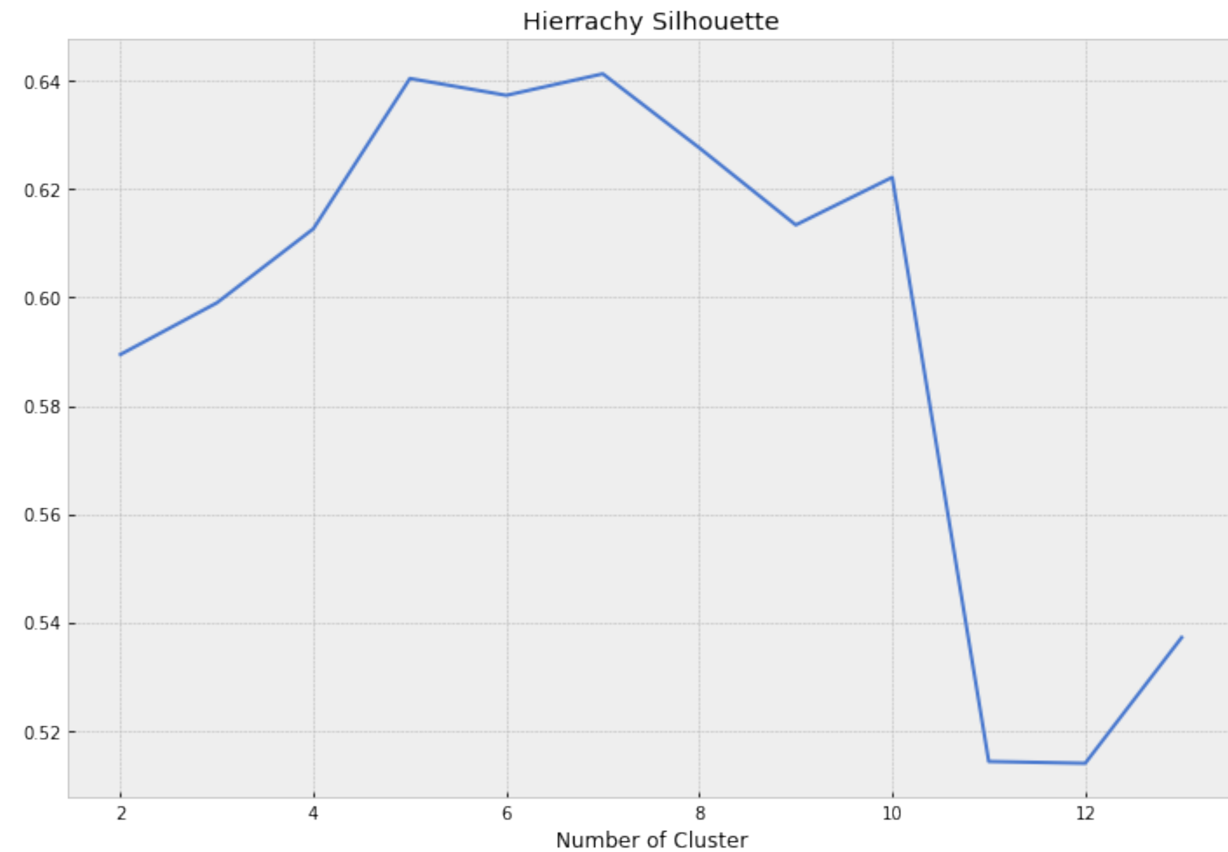


The best number of cluster is 3, which has the highest Silhouette Coefficient Score (0.661)

Hierachical Clustering

In [ ]:
```
#Apply hierarchical clustering with the evaluation based on Silhouette
Score
%%time
sil_agg = np.arange(12,dtype="double")
for k in np.arange(12):
  agg = AgglomerativeClustering(linkage='complete',
                                affinity='cosine',
                                n_clusters=k+2)
  agg.fit(X_stan)
  sil_agg[k] = metrics.silhouette_score(X_stan,agg.labels_,metric='euc
lidean')

print(sil_agg)

plt.title("Hierrachy Silhouette")
plt.xlabel("Number of Cluster")
plt.plot(np.arange(2,14,1),sil_agg)
```

```
[0.58947263 0.59902089 0.61270305 0.64045433 0.63734272 0.64131635
 0.62763897 0.61339725 0.62219949 0.51428393 0.51395113 0.53719486]
CPU times: user 18.6 s, sys: 39.5 s, total: 58.1 s
Wall time: 9.8 s
```



The best number of cluster is 6, which has the highest Silhouette Coefficient Score (0.6413)

DBSCAN

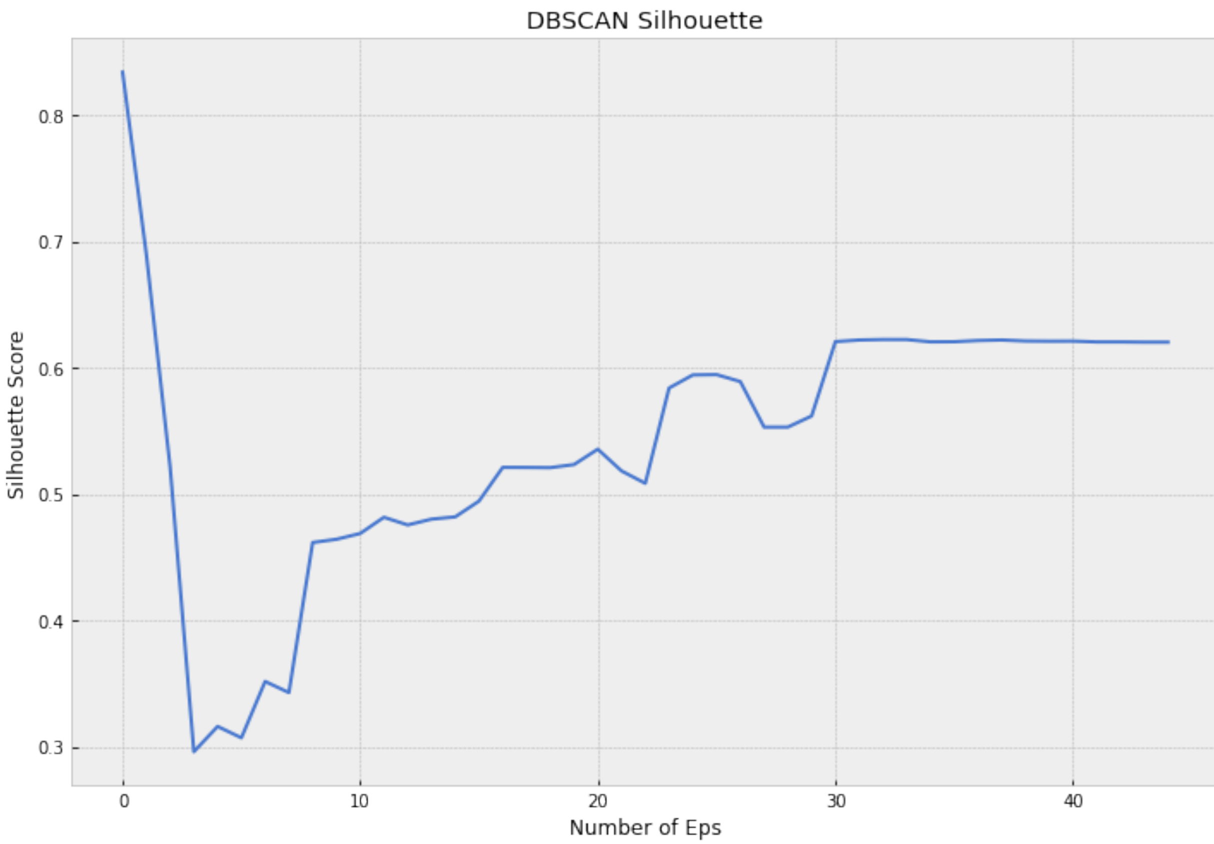
In [ ]:
```
#Apply dbscan clustering with the evaluation based on Silhouette Score
%%time
start   = 0.01
stop    = 1.5
step    = 0.01
my_list = np.arange(start, stop+step, step)

startb   = 1
stopb    = 10
stepb    = .2 # To scale proportionately with epsilon increments
my_listb = np.arange(startb, stopb+stepb, stepb)

my_range = range(45)

one = []

for i in my_range:
   dbscan = DBSCAN(eps = 0 + my_list[i] , min_samples = 0 + my_listb[i
],metric='euclidean')
   cluster = dbscan.fit_predict(X_stan)
   one.append(metrics.silhouette_score(X_stan, cluster))
```

```
CPU times: user 1min 2s, sys: 2min 27s, total: 3min 29s
Wall time: 29.1 s
```

In [ ]:
```
print(one)
plt.title("DBSCAN Silhouette")
plt.ylabel("Silhouette Score")
plt.xlabel("Number of Eps")
plt.plot(np.arange(0,45,1),one)
```

```
[0.83390272789514, 0.6896974675712358, 0.5214899828239498, 0.2960009
996743741, 0.31602024974383985, 0.3070039614776745, 0.35163051041357
55, 0.34280217812530184, 0.46160926067713237, 0.4641747860722524, 0.
46874796726878, 0.48159933401879657, 0.47550376272034955, 0.48010819
518486847, 0.48190730372026064, 0.49439577837787385, 0.5210988942708
602, 0.5210400487609188, 0.5208846186739282, 0.5233141138775801, 0.5
354853010570572, 0.5182756238180735, 0.5084571213029639, 0.583882296
0905686, 0.5942739431092124, 0.594522319445008, 0.5890126996479959,
0.5529796685562441, 0.5529796685562441, 0.5617557628345392, 0.620619
49141308, 0.6219239912666396, 0.6222976996476587, 0.6222848298078102
, 0.6204625958015156, 0.6205451016160114, 0.6215082546471166, 0.6219
320903271278, 0.6211259344061758, 0.620970111285205, 0.6210324295195
991, 0.6204010333830593, 0.6204010333830593, 0.6202330782929905, 0.6
202330782929905]
```

Out[ ]: [<matplotlib.lines.Line2D at 0x7f3c6069dc50>]



The best number of Eps is 0.01 and min sample is 1, which has the highest Silhouette Coefficient Score (0.8339)

Evaluation

In [ ]:
```
random_state = 0
dbscan_eva = DBSCAN(eps = 0.01 , min_samples = 1,metric='euclidean')
cluster_eva = dbscan.fit(X_stan)
labels = cluster_eva.labels_
```

In [ ]:
```
#print out number of cluster
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print('Estimated number of clusters: %d' % n_clusters_)
```

```
Estimated number of clusters: 2
```

Overal, the best clustering algorithms for clustering the fatilities is DBSCAN with eps = 0.01, min sample is 1, and metric is euclidean, which has 2 clusters

Visualization with dimensionality reduciton

PCA's

In [ ]:
```
# We just want the first two principal components
pca = PCA(n_components=2)

# We get the components by
# calling fit_transform method with our data
X_pca = pca.fit_transform(X)
```

In [ ]:
```
t = np.arange(5268)
plt.figure(figsize=(10,5))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=t, cmap= cm.rainbow)
plt.xticks([])
plt.yticks([])
plt.axis('off')
```

Out[ ]:
```
(-657.2833065426873, 4064.8617120360936, -76.1727773996758, 855.3370
068736015)
```

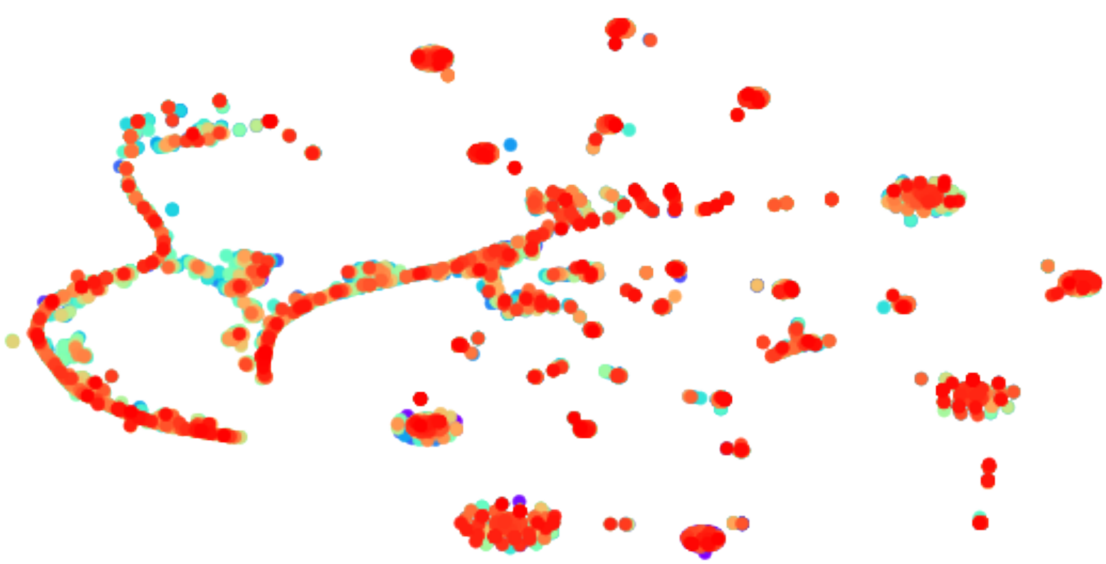
t-SNE

In [ ]:
```
%%time
tsne = TSNE(n_components=2, verbose=1, perplexity=50, n_iter=1000)
tsne_results = tsne.fit_transform(X_pca)
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 5268 samples in 0.002s...
[t-SNE] Computed neighbors for 5268 samples in 0.137s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5268
[t-SNE] Computed conditional probabilities for sample 2000 / 5268
[t-SNE] Computed conditional probabilities for sample 3000 / 5268
[t-SNE] Computed conditional probabilities for sample 4000 / 5268
[t-SNE] Computed conditional probabilities for sample 5000 / 5268
[t-SNE] Computed conditional probabilities for sample 5268 / 5268
[t-SNE] Mean sigma: 0.000000
[t-SNE] KL divergence after 250 iterations with early exaggeration:
50.490551
[t-SNE] KL divergence after 1000 iterations: 0.381874
CPU times: user 8min 21s, sys: 678 ms, total: 8min 22s
Wall time: 13.6 s
```

In [ ]:
```
plt.figure(figsize=(10,5))
plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c=t, cmap= cm.rain
bow)
plt.xticks([])
plt.yticks([])
plt.axis('off')
```

Out[ ]:
```
(-56.09588508605957,
 59.522925186157224,
 -57.439195442199704,
 62.727315711975095)
```



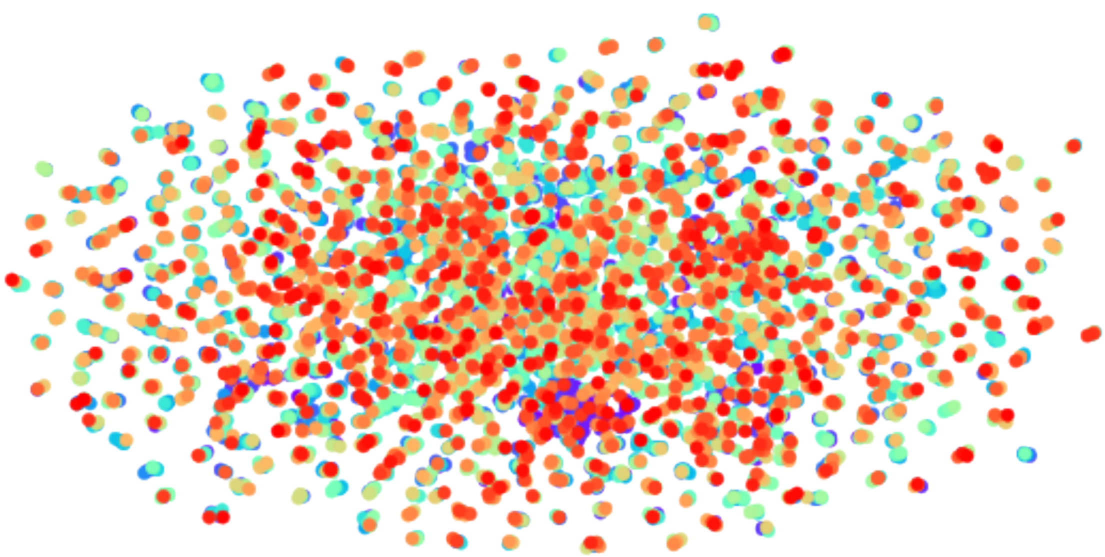
UMAP

In [ ]:
```
%%time
umap_results = umap.UMAP(n_neighbors=3, min_dist=1,
                      metric='euclidean', target_metric='categorical',
                      target_n_neighbors=-1, target_weight=0.5,
     transform_queue_size=4.0, unique=False, verbose=False).fit_transf
orm(X)
```

```
CPU times: user 1min 57s, sys: 23.2 s, total: 2min 21s
Wall time: 23 s
```

In [ ]:
```
plt.figure(figsize=(10,5))
plt.scatter(umap_results[:, 0], umap_results[:, 1], c=t, cmap= cm.rain
bow )
plt.xticks([])
plt.yticks([])
plt.axis('off')
```

Out[ ]:
```
(-17.81074423789978,
 38.07942957878113,
 -27.180909729003908,
 28.257077789306642)
```



Evaluation

> t-SNE's solution is better than those of PCA's, and UMAP's because the different classes are separated more clearly. But it is still not the best one to visualize this dataset

Clustering: Text Clustering

Data Preparation

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

Feature engineering

- K-Means normally works with numerical feature only, so I need to convert those world to number.
- The feature I use is TF-IDF. This statistic uses term frequency and inverse document frequency. The method TfidfVectorizer() implements the TF-IDF algorithm.

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

```
(4878, 9226)
```

Clustering

K-Means

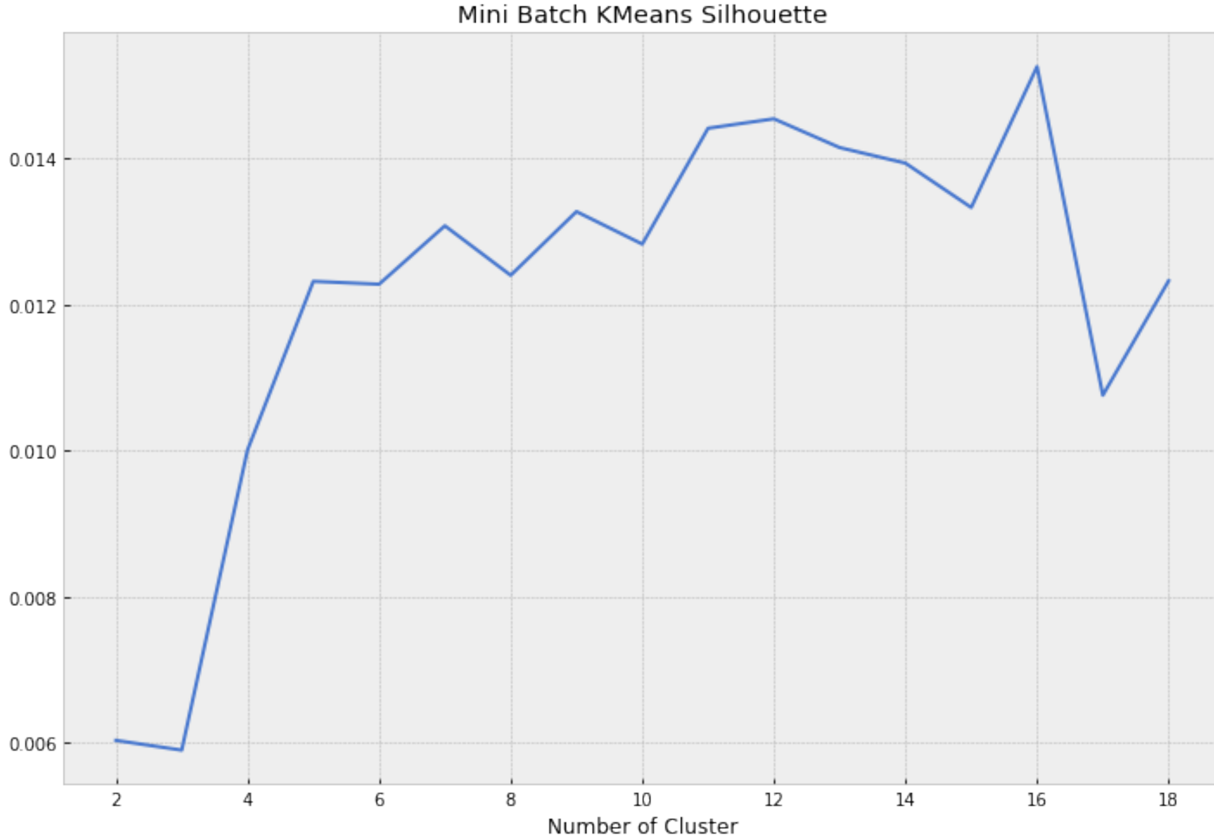
Let's test with MiniBatchKmean to get the silhouette score

MiniBatchKmean

In [ ]:

```
#Apply KMean with the evaluation based on Silhouette Score
%%time
sil_k = np.arange(17,dtype="double")
for k in np.arange(17):
  minikm = MiniBatchKMeans(n_clusters=k+2, random_state=0)
  minikm.fit(X)
  sil_k[k] = metrics.silhouette_score(X,minikm.labels_,metric='euclide
an')

print(sil_k)

plt.title("Mini Batch KMeans Silhouette")
plt.xlabel("Number of Cluster")
plt.plot(np.arange(2,19,1),sil_k)
```

```
[0.00603299 0.00589956 0.01000973 0.01231947 0.01228014 0.01307989
 0.01240237 0.01327464 0.01282931 0.01441571 0.01454498 0.01415107
 0.01393782 0.01333072 0.01525917 0.01075958 0.0123267 ]
CPU times: user 12.7 s, sys: 130 ms, total: 12.9 s
Wall time: 12.9 s
```



According to this chart, the best silhouette score is 0.0152 when number of clusters are 16

> Let's apply K-means Model

KMeans

In [ ]:
```
%%time
model = KMeans(n_clusters=16, random_state=0)
model.fit(X)
```

```
CPU times: user 3min 48s, sys: 18min 51s, total: 22min 40s
Wall time: 38.6 s
```

In [ ]:
```
%%time
print ('Most Common Terms per Cluster:')

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

for i in range(16):
    print("\n")
    print('Cluster %d:' % i)
    for j in order_centroids[i, :17]: #replace 17 with n words per clu
ster
        print ('%s' % terms[j]),
    print
```

```
Most Common Terms per Cluster:


Cluster 0:
crashed
plane
aircraft
killed
flight
miles
air
fuel
helicopter
midair
aboard
collision
airport
mountains
taking
pilot
minutes


Cluster 1:
pilot
mountain
terrain
error
flight
crashed
weather
altitude
ft
fog
```

```
aircraft
conditions
poor
high
low
struck
crew


Cluster 2:
takeoff
crashed
shortly
engine
overloaded
failure
exploded
wing
ocean
mountain
losing
aircraft
plane
cashed
building
river
aborted


Cluster 3:
weather
conditions
adverse
vfr
continued
flight
poor
crashed
pilot
mountain
route
ifr
en
related
flew
fog
low


Cluster 4:
```

```
mountain
struck
flew
crashed
poor
weather
000
cargo
plane
ft
fog
positioning
visibility
approach
conditons
descending
conditions


Cluster 5:
sea
crashed
helicopter
ditched
poor
mediterranean
approach
cause
weather
conditions
unknown
miles
lost
takeoff
recovered
runway
taking


Cluster 6:
taking
shortly
crashed
plane
engine
airport
aircraft
failure
cargo
unknown
```

```
mountain
mail
minutes
air
lost
ocean
area


Cluster 7:
attempting
land
crashed
plane
runway
cargo
fog
struck
trees
short
burned
poor
heavy
airport
pilot
weather
visibility


Cluster 8:
shot
rebels
missile
air
surface
british
aircraft
forces
anti
military
afghan
japanese
enemy
rebel
fighters
unita
fighter


Cluster 9:
```

```
engine
failure
takeoff
crashed
aircraft
experiencing
plane
failed
lost
right
emergency
left
airport
power
fuel
return
pilot


Cluster 10:
approach
crashed
runway
short
aircraft
final
crew
pilot
fog
trees
descent
instrument
ground
ils
visual
miles
altitude


Cluster 11:
landing
runway
crashed
attempt
emergency
plane
gear
aircraft
attempting
make
```

```
pilot
caught
crew
struck
short
engine
forced


Cluster 12:
aircraft
control
crashed
runway
loss
pilot
takeoff
flight
plane
failure
ground
left
wing
crew
lost
right
altitude


Cluster 13:
en
route
crashed
disappeared
mountain
plane
cargo
mountains
mountainous
went
missing
struck
wreckage
terrain
aircraft
poor
undetermined


Cluster 14:
```

```
flames
burst
plane
crashed
aircraft
engine
runway
taking
landing
bursting
takeoff
hit
shortly
field
struck
landed
failed


Cluster 15:
cargo
plane
crashed
runway
struck
takeoff
engine
lost
approach
altitude
mountain
trees
attempting
ocean
shifted
gain
short
CPU times: user 192 ms, sys: 989 ms, total: 1.18 s
Wall time: 29.9 ms
```

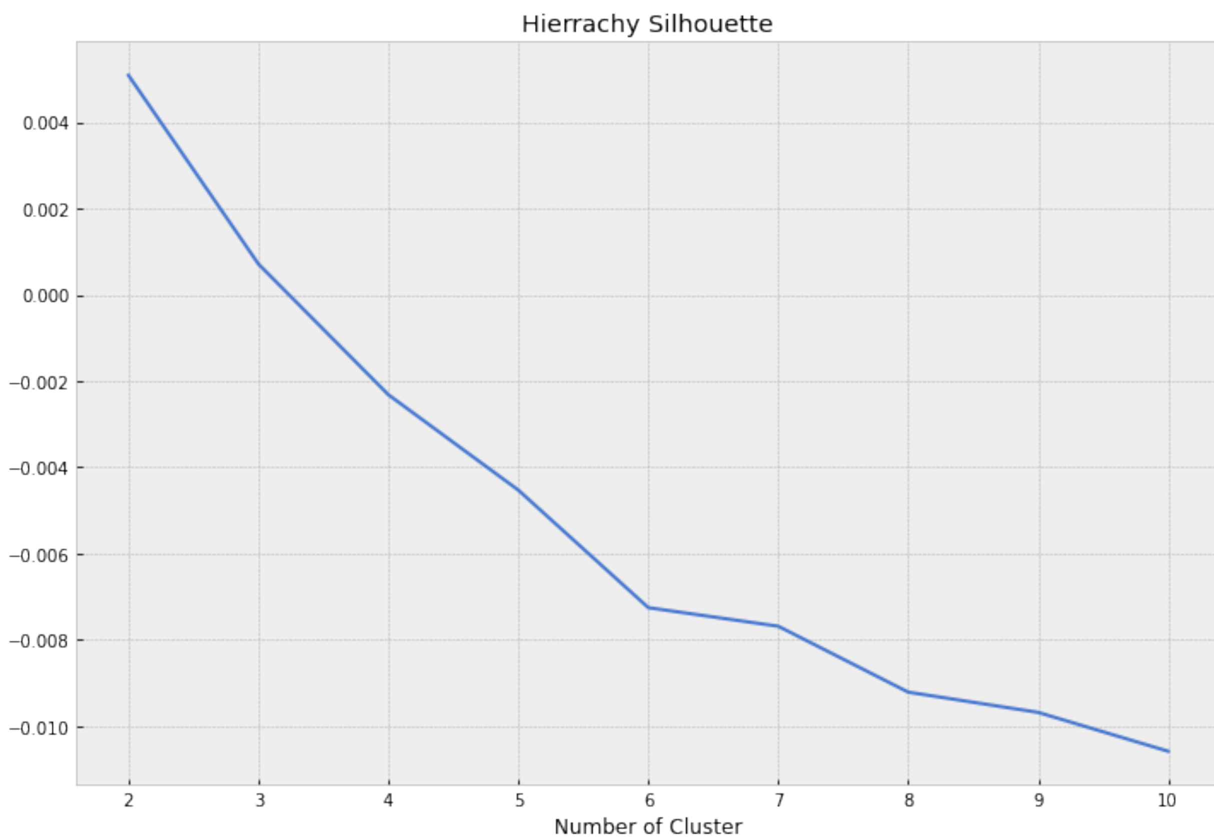
Hierarchical Clustering

In [ ]:

```
X1=X.todense()
```

In [ ]:
```
#Apply hierarchical clustering with the evaluation based on Silhouette
Score
%%time
sil_agg = np.arange(9,dtype="double")
for k in np.arange(9):
  agg = AgglomerativeClustering(linkage='complete',
                                  affinity='cosine',
                                  n_clusters=k+2)
  agg.fit(X1)
  sil_agg[k] = metrics.silhouette_score(X1,agg.labels_,metric='euclide
an')

print(sil_agg)

plt.title("Hierrachy Silhouette")
plt.xlabel("Number of Cluster")
plt.plot(np.arange(2,11,1),sil_agg)
```

```
[ 0.00509052  0.0007139  -0.00231751 -0.00452922 -0.00725523 -0.0076
8596
 -0.00921466 -0.00968406 -0.01058528]
CPU times: user 17min 38s, sys: 1min 5s, total: 18min 43s
Wall time: 15min 44s
```



The chart show the best number of cluster for Hierarchy is 2. However, it's not the good clustering method in this case.

DBSCAN

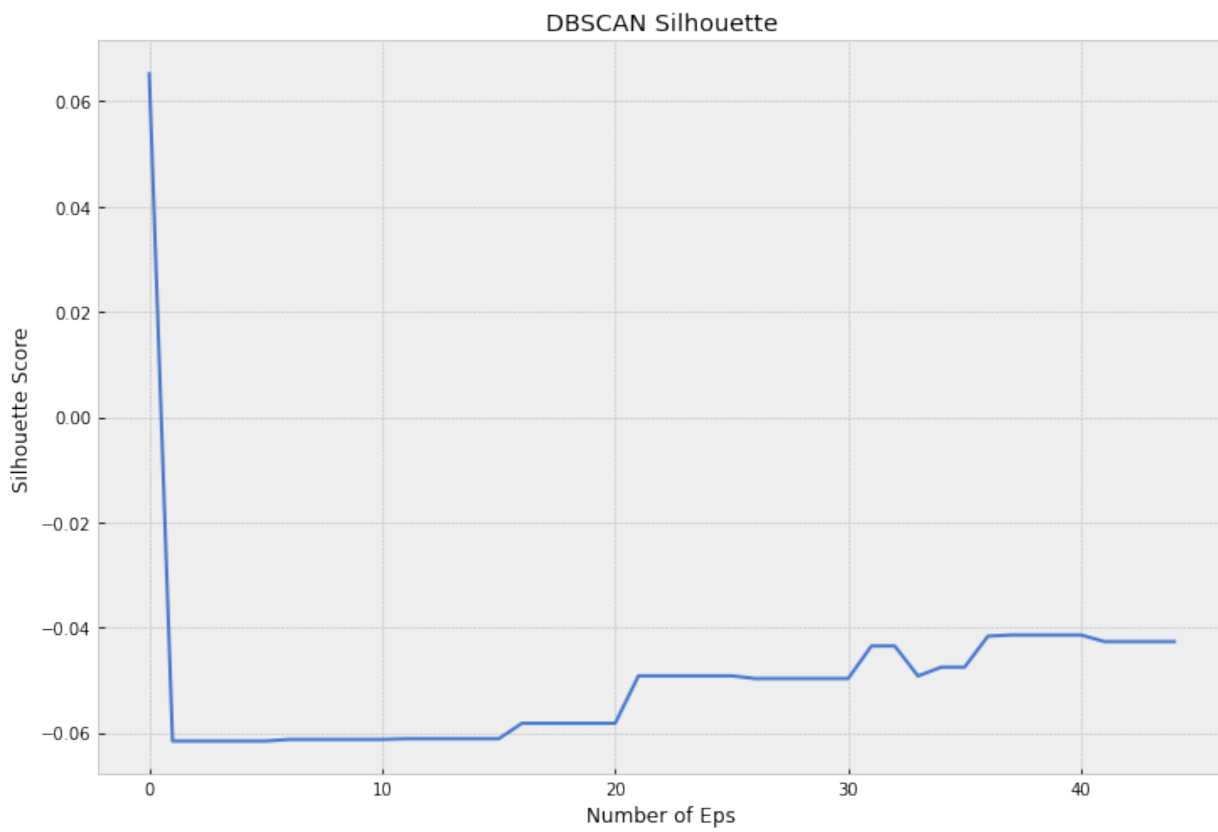
In [ ]:
```
#Apply dbscan clustering with the evaluation based on Silhouette Score
%%time
start   = 0.01
stop    = 1.5
step    = 0.01
my_list = np.arange(start, stop+step, step)

startb   = 1
stopb    = 10
stepb    = .2 # To scale proportionately with epsilon increments
my_listb = np.arange(startb, stopb+stepb, stepb)

my_range = range(45)

one = []

for i in my_range:
   dbscan = DBSCAN(eps = 0 + my_list[i] , min_samples = 0 + my_listb[i
],metric='euclidean')
   cluster = dbscan.fit_predict(X1)
   one.append(metrics.silhouette_score(X1, cluster))
```

```
CPU times: user 38min 11s, sys: 5min 14s, total: 43min 25s
Wall time: 28min 35s
```

In [ ]:
```
print(one)
plt.title("DBSCAN Silhouette")
plt.ylabel("Silhouette Score")
plt.xlabel("Number of Eps")
plt.plot(np.arange(0,45,1),one)
```

```
[0.06519065190651907, -0.06155053044562705, -0.06155053044562705, -0
.06155053044562705, -0.06155053044562705, -0.06155053044562705, -0.0
6124606444412675, -0.06124606444412675, -0.06124606444412675, -0.061
24606444412675, -0.06124606444412675, -0.06112707664086779, -0.06112
707664086779, -0.06112707664086779, -0.06112707664086779, -0.0611270
7664086779, -0.05818283552484629, -0.05818283552484629, -0.058182835
52484629, -0.05818283552484629, -0.05818283552484629, -0.04916645903
007011, -0.04916645903007011, -0.04916645903007011, -0.0491664590300
7011, -0.04916645903007011, -0.049681402067601, -0.049681402067601,
-0.049681402067601, -0.049681402067601, -0.049681402067601, -0.04349
383000199573, -0.04349383000199573, -0.04919711006233494 5, -0.047515
0457232907, -0.0475150457232907, -0.04162064645892075, -0.0414059084
4087734, -0.04140590844087734, -0.04140590844087734, -0.041405908440
87734, -0.04265825287295191, -0.04265825287295191, -0.04265825287295
191, -0.04265825287295191]
```

Out[ ]: [<matplotlib.lines.Line2D at 0x7f3c5ccead68>]



Evaluation

In [ ]:
```
random_state = 0
dbscan_eva = DBSCAN(eps = 0.01 , min_samples = 1,metric='euclidean')
cluster_eva = dbscan_eva.fit(X1)
labels = cluster_eva.labels_
```

In [ ]:
```
#print out number of cluster
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print('Estimated number of clusters: %d' % n_clusters_)
```

```
Estimated number of clusters: 4627
```

The best model which has the highest Silhoette Coefficient Score (0.065) is DBSCAN with eps = 0.01, min_sample = 1, and metric etric='euclidean' with the number of cluster is 4627

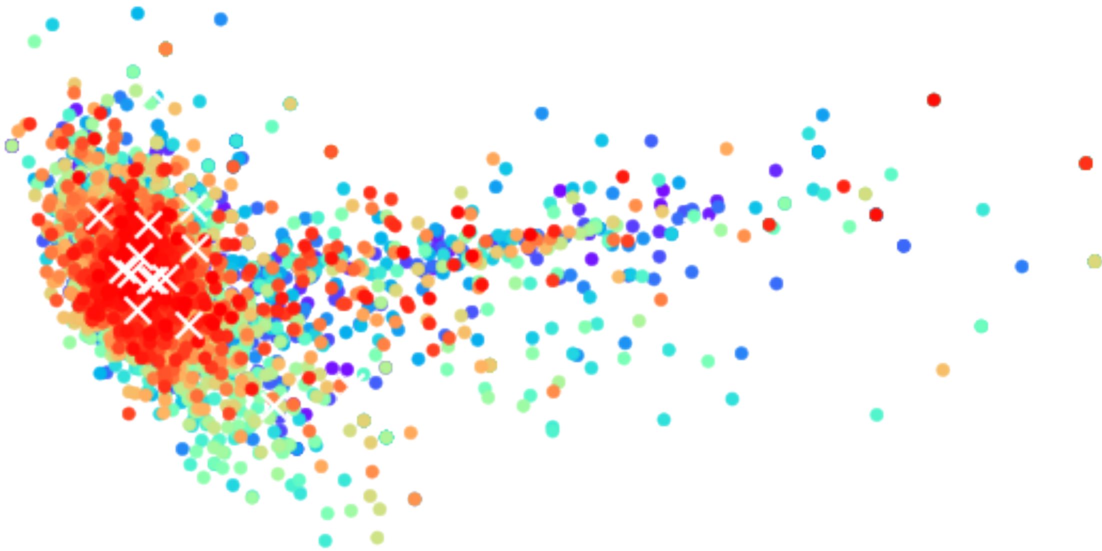
Visualize clustering

PCA

In [ ]:
```
# We just want the first two principal components
pca = PCA(n_components=2)

# We get the components by
# calling fit_transform method with our data
pca_components = pca.fit_transform(X.toarray())

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

In [ ]:
```
t = np.arange(4878)
plt.figure(figsize=(10,5))
plt.scatter(pca_components[:, 0], pca_components[:, 1], c=t, cmap= cm.
rainbow)
plt.scatter(reduced_cluster_centers[:, 0], reduced_cluster_centers[:,1
], marker='x', s=150, c='w')
plt.xticks([])
plt.yticks([])
plt.axis('off')
```

Out[ ]:
```
(-0.21520871657518625,
 0.9230199694686285,
 -0.5930245771164822,
 0.5949421005253328)
```



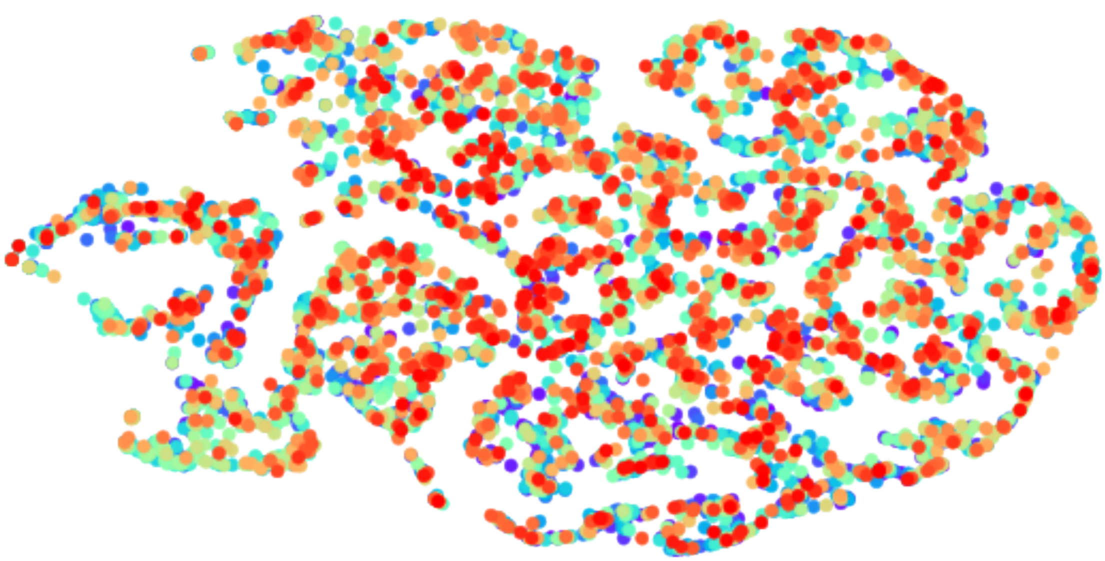
t-SNE

In [ ]:
```
%%time
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
tsne_results = tsne.fit_transform(pca_components)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 4878 samples in 0.002s...
[t-SNE] Computed neighbors for 4878 samples in 0.104s...
[t-SNE] Computed conditional probabilities for sample 1000 / 4878
[t-SNE] Computed conditional probabilities for sample 2000 / 4878
[t-SNE] Computed conditional probabilities for sample 3000 / 4878
[t-SNE] Computed conditional probabilities for sample 4000 / 4878
[t-SNE] Computed conditional probabilities for sample 4878 / 4878
[t-SNE] Mean sigma: 0.005184
[t-SNE] KL divergence after 250 iterations with early exaggeration:
66.464859
[t-SNE] KL divergence after 300 iterations: 1.432712
CPU times: user 2min 6s, sys: 250 ms, total: 2min 6s
Wall time: 3.74 s
```

In [ ]:
```
plt.figure(figsize=(10,5))
plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c=t, cmap= cm.rain
bow)
plt.xticks([])
plt.yticks([])
plt.axis('off')
```

Out[ ]:
```
(-19.515973091112549,
 17.440865516662598,
 -16.204239559173583,
 15.264069271087646)
```



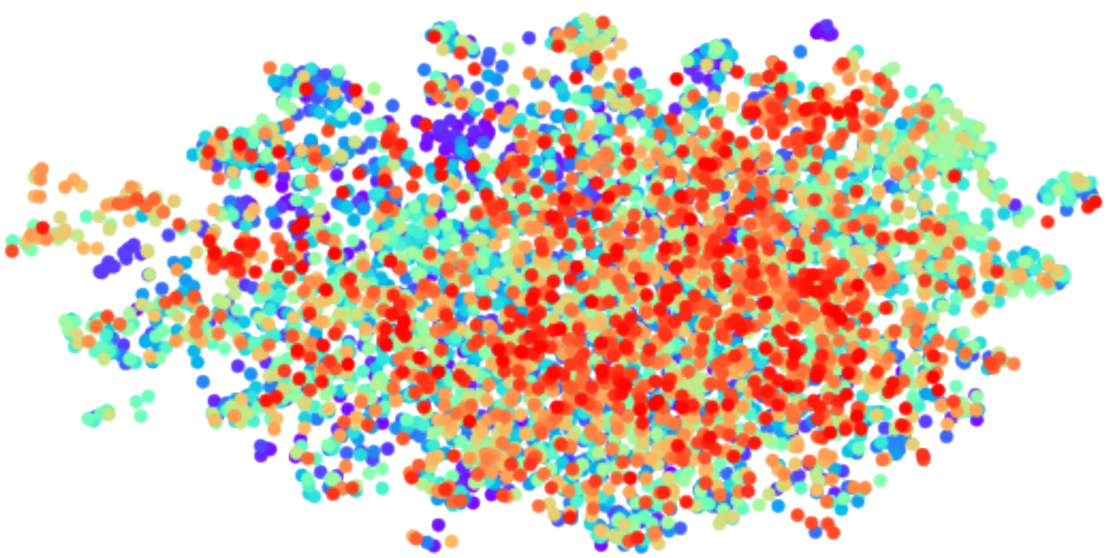
UMAP

In [ ]:
```
%%time
umap_results = umap.UMAP(n_neighbors=9, min_dist=1,
                      metric='euclidean', target_metric='categorical',
                      target_n_neighbors=-1, target_weight=0.5,
     transform_queue_size=4.0, unique=False, verbose=False).fit_transf
orm(X.toarray())
```

```
CPU times: user 2min 36s, sys: 6.42 s, total: 2min 42s
Wall time: 11.5 s
```

In [ ]:
```
plt.figure(figsize=(10,5))
plt.scatter(umap_results[:, 0], umap_results[:, 1], c=t, cmap= cm.rain
bow )
plt.xticks([])
plt.yticks([])
plt.axis('off')
```

Out[ ]:
```
(-0.2138025581836701, 19.02335940003395, -10.600976777076 72, 7.85481
1024665833)
```



Evaluation

> There is no solution better than others. PCA's, t-SNE's, and UMAP's cannot show the different classes which are not separated more clearly.

Conclusion based on this research

1. Airplane Crashing

- USA has the highest fatility cases in the world
- There are over 85% of airplan crash is from commercial flights
- Don't fly with Aeroflot Operator, there is 68% chance of you dying.
- Don't fly in a Douglas DC-3 aircraft, you are 5 times more chance to die.
- Don't take any flight that flies Tenerife - Las Palmas / Tenerife - Las Palmas route or Tokyo - Osaka.
- Don't fly with any flight of type Douglous DC-3 ,it had highest fatalities percentage.
- Avoid going to Sao Paulo, Brazil ; Moscow, Russia ; Rio de Janeiro, Brazil ; they had highest plane crash location.
- It is so much safer to take flight now-a-days as compared to 1970-80, 1972 was the worst years for airline industry.
- Peole who are born in year of Ox have more chance to die, and people who are born in year of Horse have high chance to survive
- The best clustering algorithms for clustering the fatilities is DBSCAN with eps = 0.01, min sample is 1, and metric is euclidean with the number of cluster is 2

1. Text Clustering

- It's hard to determine which is the best method. Overall, The best model which has the highest Silhoette Coefficient Score (0.065) is DBSCAN with eps = 0.01, min_sample = 1, and metric etric='euclidean', which has 4627 clusters
- Personally, I like K-mean because it returns 16 cluster, which is easy to understand and visuallize.

1. Outcome

- There is no absolute right answer for clustering. To bring the best results for business solution, I would recomend data scientists consider the best model that fit the business need and apply their business acumen to suggest the best strategy for the whole team.
- My data analysis above is good for travelers around the world to have an insider about traviling with airplane. And it is a good suggestion for operators or aerospace company to look up the disavantages of the past to improve their future businesses.