internship

August 12, 2024

A csv file containing the ship movement data is given. The data contains unique ship id (mmid), timestamp, latitude and longitude.

The task given is to find probable collision events

Exploratory data analysis and plotting the data onto world map.

[140]:
```
#importing the libraries  and defining the haversine distance function
import numpy as np
import geopandas as gd
import pandas as pd
from geodatasets import get_path
import matplotlib.pyplot as plt
from sklearn.neighbors import BallTree
from sklearn.cluster import DBSCAN
from shapely.geometry import LineString, Point
from math import radians, cos, sin, asin, sqrt
```

Haversine distance is the distance between two points on a sphere.

$d = 2R\sin^{-1}\left(\sqrt{(\sin^2\left(\frac{\phi_2-\phi_1}{2}\right)\cos\phi_1\cos\phi_2\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right)$

where R is the radius of the sphere, $\phi_{1,2}$ are latitudes and $\lambda_{1,2}$ are the longitudes

[141]:
```
def haversine_np(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

    # haversine formula
    dlon = lon2 - lon1[:,None]
    dlat = lat2 - lat1[:,None]
    a = np.sin(dlat/2)**2 + np.cos(lat1[:,None]) * np.cos(lat2) * np.sin(dlon/
↪2)**2
    #print("a=",a)
    c = 2 * np.arcsin(np.sqrt(a))
```

```
    r = 6371 # Radius of earth: 6371 kilometers
    return c * r
```

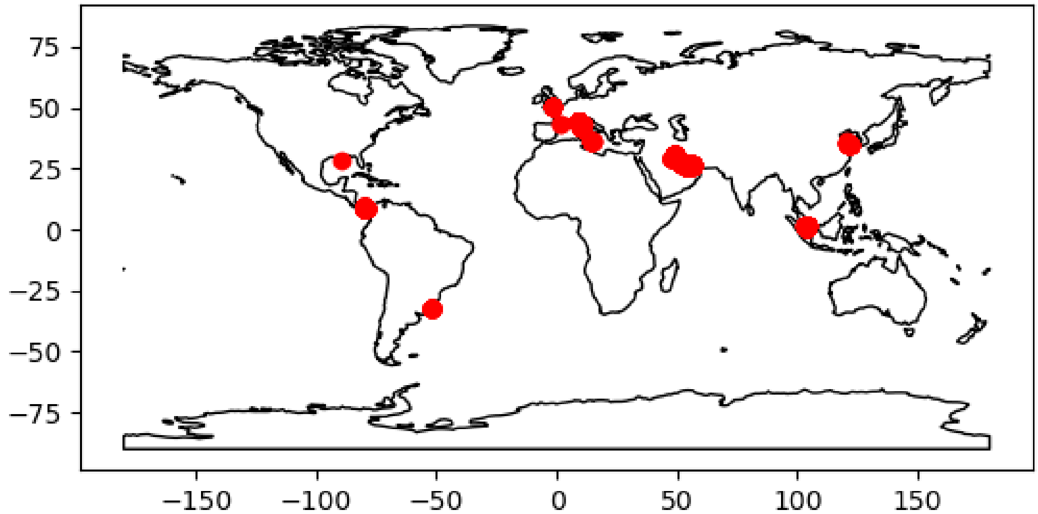
Reading the data into pandas dataframe for exploratory analysis

[142]:
```
df = pd.read_csv("d:\sample_data.csv")
print(df.info())
print(df.head())
#converting go geodataframe
gdf = gd.GeoDataFrame(df, geometry=gd.points_from_xy(df.lon, df.lat), crs="EPSG:
↪4326")
#getting the world map
world = gd.read_file(get_path("naturalearth.land"))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13501 entries, 0 to 13500
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   mmsi       13501 non-null  int64
 1   timestamp  13501 non-null  object
 2   lat        13501 non-null  float64
 3   lon        13501 non-null  float64
dtypes: float64(2), int64(1), object(1)
memory usage: 422.0+ KB
None
        mmsi                 timestamp       lat        lon
0  565761000  2023-03-15 00:27:44+00   1.26878  103.75827
1  538008084  2023-03-19 23:30:00+00  43.55962   10.29404
2  564654000  2023-03-12 08:22:53+00   1.23725  103.89135
3  529123000  2023-03-05 16:47:42+00  29.44367   48.93066
4  564780000  2023-03-11 06:35:20+00   1.27755  103.61026
```

[143]:
```
#plotting the world map as background
ax = world.plot(color="white", edgecolor="black")
# plotting the geodatframe.
gdf.plot(ax=ax, color="red")
```

[143]: <AxesSubplot:>



Each point in the plot above is the location of ship covering all times. There are multiple points overlapping.

[144]:
```
#convert timestamp into datetime format and get time difference.

df['timestamp']=pd.to_datetime(df['timestamp'])
start_time=df['timestamp'].min()

df['timediff']=(df['timestamp']-start_time).astype("timedelta64[s]")/60   #in␣
↪minutes
```

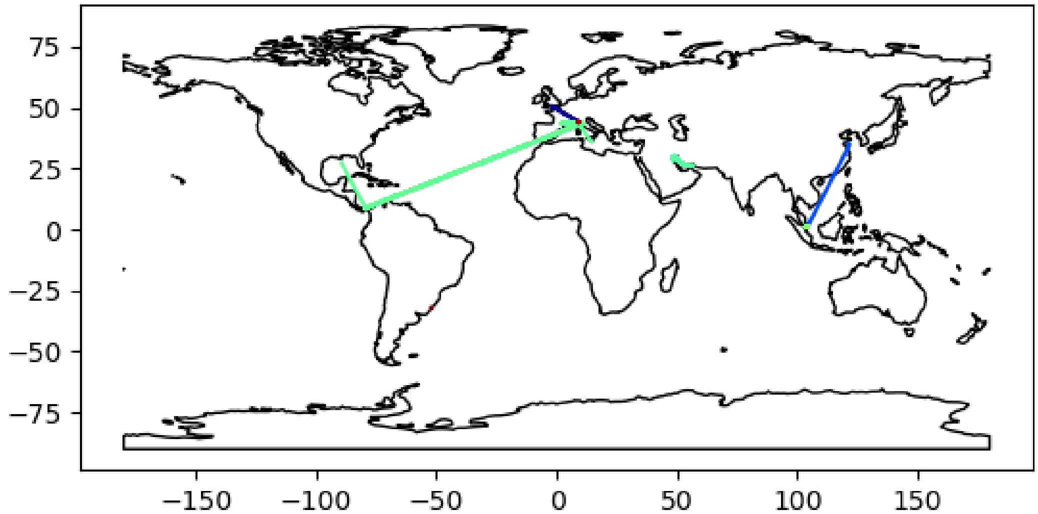
[145]:
```
df['geometry']=df.apply(lambda x: Point(x['lon'],x['lat']),axis=1)
```

[146]:
```
s_df = df.sort_values(by=["mmsi","timestamp"],ascending=[True,False])
```

[147]:
```
linedf = s_df.groupby("mmsi", as_index=False).agg({"geometry": lambda x:␣
↪LineString(x)})
```

[148]:
```
#plotting the movement of each ship as line
linedf = gd.GeoDataFrame(data=linedf, geometry=linedf.geometry, crs=4326)
ax = world.plot(color="white", edgecolor="black")

linedf.plot(ax=ax, column="mmsi", cmap="jet")
plt.show()
```



The plot above is a line representation of ship-data. each color represent one line, resulting in a timeseries of different ships.

Potential collision is when two different ships come in close proximity within a small time window. To analyze this, first we will separate each ship's own timeline. Then, within a small window around each timestamp, we will analyze whether there is any other ship within a cutoff distance.

We have taken time window of ±1 minute, and window of ±0.01° around latitude and longitude. We further refined it with a distance of ±1 kilometer.

[149]:
```
#get number of ships
mmsi_a=df['mmsi'].unique()
```

[150]:
```
print(mmsi_a)
```

```
[565761000 538008084 564654000 529123000 564780000 563014650 563078430
 352656000 538008064 518998309 875832716 232006548 898998797 232345740
 352002300 218719092 889799564]
```

[173]:
```
df_final=pd.DataFrame()   #This is the dataframe to collate all the possible␣
 ↪collision events
for i in mmsi_a:
    #Getting the timeseries for ship i
    timeseries_df=df[df['mmsi']==i]

    timeseries=timeseries_df['timediff'].unique()

    #Getting data for all ships excluding ship i
    df_new=df[df['mmsi']!=i]
```

```
    for j in timeseries:

        #Recording the lat lon for ship i at time instance j. This will be␣
↪useful for distance calculation
        orig_lat=pd.
↪to_numeric(timeseries_df[timeseries_df['timediff']==j]['lat'].unique())
        orig_lon=pd.
↪to_numeric(timeseries_df[timeseries_df['timediff']==j]['lon'].unique())

        #Filtering the data frame based on time window
        df_new4=df_new[abs(df_new['timediff']-j)<1]
        #Further filtering based on lat-lon
        df_new3=df_new4[abs(df_new4['lat']-orig_lat)<0.01]
        df_new2=df_new3[abs(df_new3['lon']-orig_lon)<0.01]

        if df_new2.empty:
            continue

        d=haversine_np(orig_lon,orig_lat, pd.to_numeric(df_new2['lon']).
↪values,pd.to_numeric(df_new2['lat']).values).ravel()

        #Adding distance column, and final filter based on distance window

        df_new2.insert(6,'dist',d)
        df_new5=df_new2[df_new2['dist']<1]
        #df_new5['mmsi_1']=i   #The collision event is between ship i and ship␣
↪mmid in df_new5
        df_new5.insert(7,'mmsi_1',i)
        #Appending the entry to the df_final dataframe
        if not(df_new5.empty):
            df_final=pd.concat([df_final,df_new5],ignore_index=True)

        else:
           continue
```

[174]:
```
print(df_final.head())
```

```
        mmsi                  timestamp       lat         lon      timediff  \
0  564780000 2023-03-10 19:30:00+00:00  1.227305  103.717684   8983.783333
1  563014650 2023-03-19 08:30:00+00:00  1.299828  103.956321  21283.783333
2  563014650 2023-03-19 11:30:00+00:00  1.300234  103.955836  21463.783333
3  352002300 2023-03-12 15:30:00+00:00  1.249132  103.930341  11623.783333
4  563078430 2023-03-20 21:30:00+00:00  1.224983  103.716907  23503.783333
```

```
                                          geometry      dist     mmsi_1
0  POINT (103.7176839627682 1.227304904255787)  0.910299  565761000
1  POINT (103.9563211059508 1.299828096172364)  0.582343  565761000
2   POINT (103.955836186487 1.300233952693354)  0.519866  565761000
3  POINT (103.9303411553958 1.249132441479197)  0.067017  565761000
4  POINT (103.7169065780942 1.224983259883276)  0.516348  565761000
```

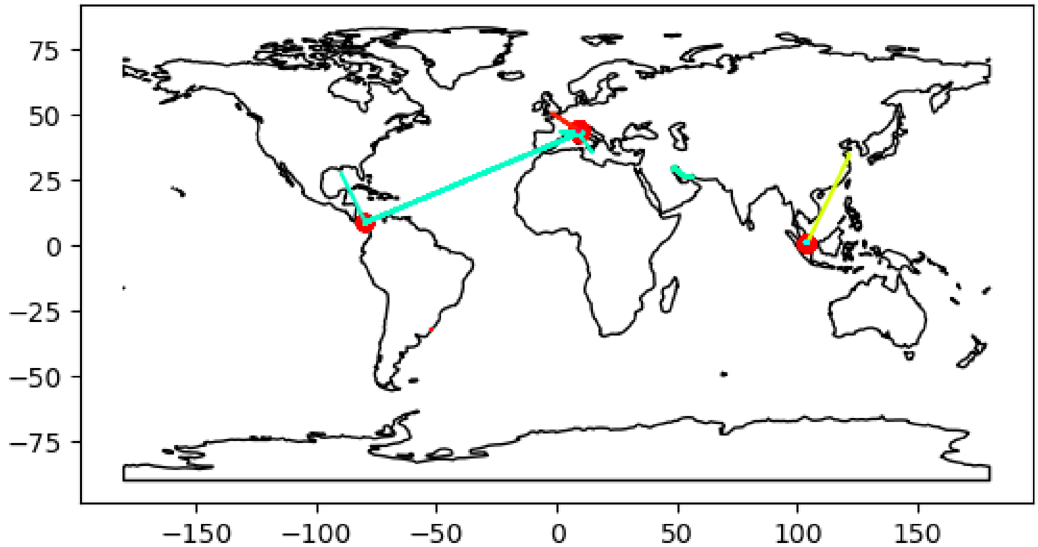
[175]:
```python
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1276 entries, 0 to 1275
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   mmsi       1276 non-null   int64
 1   timestamp  1276 non-null   datetime64[ns, UTC]
 2   lat        1276 non-null   float64
 3   lon        1276 non-null   float64
 4   timediff   1276 non-null   float64
 5   geometry   1276 non-null   object
 6   dist       1276 non-null   float64
 7   mmsi_1     1276 non-null   int64
dtypes: datetime64[ns, UTC](1), float64(4), int64(2), object(1)
memory usage: 79.9+ KB
```

The number of potential collision events can be obtained from the info of the dataframe. For the conditions set-up here, there are 1276 collision events. The plot below overlays the events as points on the lines of each ships.

[164]:
```python
gdf = gd.GeoDataFrame(df_final, geometry=gd.points_from_xy(df_final.lon,␣
 ↪df_final.lat), crs="EPSG:4326")
ax = world.plot(color="white", edgecolor="black")
linedf.plot(ax=ax, column="mmsi", cmap="hsv")
gdf.plot(ax=ax, color="red")
```

[164]: <AxesSubplot:>


75
50
25
0
−25
−50
−75
−150
−100
−50
0
50
100
150

[ ]: