Predictive analysis of naval incidents in the USA, 2002 - 2015:

Annex 3.1. Preprocess Casualty & Pollution

Author: Oscar Anton
Date: 2024

Version: 0.9

0. Loadings

Libraries

In [2]:
```
# General data management
import numpy as np
import pandas as pd

# Geopositioning data management
import geopandas as gpd
from shapely.geometry import Point

# Visualization
import plotly.graph_objects as go
```

General variables

In [3]:
```
# Main data folders
import_data_folder = 'RawDataAllCasualtyAndPollution'
export_data_folder = 'DataCasualtyAndPollution'

# Maps data folder
maps_folder = 'Maps'

# Toggle for export data to external file
file_export_enabled = False
```

1. Data Acquisition

In [4]:
```
# Read text file and creating a DataFrame for variable names
variableNames = pd.read_csv(import_data_folder + '/' + 'VariableNames.txt',
                            delimiter='\t',
                            encoding='ISO-8859-1')

# Save to external file
if file_export_enabled :
    variableNames.reset_index().to_feather(import_data_folder + '/' + 'VariableNames.feather')
    print(f'variableNames {variableNames.shape} exported to {import_data_folder}')
else:
    variableNames = pd.read_feather(import_data_folder + '/' + 'VariableNames.feather')
    print(f'variableNames {variableNames.shape} imported from {import_data_folder}')
```

```
variableNames (276, 6) imported from RawDataAllCasualtyAndPollution
```

2. Data delimitation

2.1. VslEvents

In [5]:
```
# Read text file and creating a DataFrame
VslEvents = pd.read_csv(import_data_folder + '/' + 'MisleVslEvents.txt',
                        delimiter='\t',
                        encoding='ISO-8859-1')

# Dataframe variable names from variableNames
VslEvents.columns = variableNames.query('Table == "MisleVslEvents"')['Name'].tolist()

# timeline_dt variable set up
VslEvents['timeline_dt'] = pd.to_datetime(VslEvents['timeline_dt'],
                                          format="%Y-%m-%d %H:%M:%S.%f")
VslEvents['date'] = VslEvents['timeline_dt']
VslEvents['hour'] = VslEvents['timeline_dt'].dt.time

# Variable preselection
VslEvents = VslEvents.drop(['timeline_dt', 'case_id', 'fk_d_vessel',
                            'vessel_service', 'vessel_type', 'vessel_subtype',
                            'event_class', 'event_subclass'], axis=1)

# Filter by coords and date
VslEvents = VslEvents[
    (VslEvents['latitude'].between(15, 70)) &
```

```
    (VslEvents['longitude'].between(-180, -45)) &
    (VslEvents['date'].between(pd.to_datetime('2002-01-01'),
                               pd.to_datetime('2015-12-31')))
]

# Distinct rows by ID variables
VslEvents = VslEvents.drop_duplicates(subset=['activity_id', 'vessel_id', 'event_type'], keep='first')

# Save to external file
if file_export_enabled :
    VslEvents.reset_index().to_feather(export_data_folder + '/' + 'VslEvents.feather')
    print(f'VslEvents {VslEvents.shape} exported to {export_data_folder}')
else:
    VslEvents = pd.read_feather(export_data_folder + '/' + 'VslEvents.feather')
    print(f'VslEvents {VslEvents.shape} imported to {export_data_folder}')
```

```
VslEvents (120433, 15) imported to DataCasualtyAndPollution
```

2.2. Vessel

In [6]:

```
# Read text file and creating a DataFrame
Vessel = pd.read_csv(import_data_folder + '/' + 'MisleVessel.txt',
                     delimiter='\t',
                     encoding='ISO-8859-1',
                     quoting=3,
                     low_memory=False,
                     usecols=range(66))

# Dataframe variable names from variableNames
Vessel.columns = variableNames.query('Table == "MisleVessel"')['Name'].tolist()

# Variable preselection
Vessel = Vessel.drop(['gk_d_vessel', 'managing_owner_id', 'managing_owner',
                      'net_ton', 'itc_breadth', 'itc_depth', 'itc_gross_ton',
                      'itc_length', 'itc_net_ton', 'draft_design', 'draft_design_units',
                      'deadweighttonnage_units', 'hailing_port', 'hailing_port_state',
                      'hailing_port_province', 'route_type', 'cargo_authorization_type',
                      'documented_ind', 'documented_status_type', 'inspected_ind',
                      'inspected_desc', 'state_vessel_ind', 'state_vessel_desc',
                      'lloyds_ind', 'lloyds_desc', 'solas_ind', 'insp_subchapter_type',
                      'vessel_type', 'vessel_subtype', 'vessel_service',
                      'max_passengers_allowed', 'max_crew', 'self_propelled_ind',
                      'call_sign', 'official_number', 'hull_number', 'rbs_hull_number',
                      'vessel_age', 'hull_build_party_name', 'completed_by_party_name',
                      'filler'], axis=1)

# Distinct rows by ID variables
Vessel = Vessel.drop_duplicates(
    subset=['vessel_id', 'vessel_name'], keep='first')

# Fix dtypes
Vessel['dead_weight_ton'] = pd.to_numeric(Vessel['dead_weight_ton'], errors='coerce')
Vessel['primary_vin'] = Vessel['primary_vin'].astype(str)
Vessel['imo_number'] = Vessel['imo_number'].astype(str)
Vessel['build_year'] = Vessel['build_year'].astype(str)
Vessel['horsepower_ahead'] = pd.to_numeric(Vessel['horsepower_ahead'], errors='coerce')
Vessel['horsepower_astern'] = pd.to_numeric(Vessel['horsepower_astern'], errors='coerce')

# Save to external file
if file_export_enabled :
    Vessel.reset_index().to_feather(export_data_folder + '/' + 'Vessel.feather')
    print(f'Vessel {Vessel.shape} exported to {export_data_folder}')
else:
    Vessel = pd.read_feather(export_data_folder + '/' + 'Vessel.feather')
    print(f'Vessel {Vessel.shape} imported to {export_data_folder}')
```

```
Vessel (1346644, 26) imported to DataCasualtyAndPollution
```

2.3. Injury

In [7]:

```
# Read text file and creating a DataFrame
Injury = pd.read_csv(import_data_folder + '/' + 'MisleInjury.txt',
                     delimiter='\t',
                     encoding='ISO-8859-1')

# Dataframe variable names from variableNames
Injury.columns = variableNames.query('Table == "MisleInjury"')['Name'].tolist()

# Variable preselection
Injury = Injury.drop(['fk_d_vessel', 'vessel_service', 'vessel_type',
                      'vessel_subtype', 'facility_id', 'facility_name',
                      'facility_type_desc', 'facility_activity_role_desc'],
                     axis=1)

# Filter by coords
Injury = Injury[
    (Injury['latitude'].between(15, 70)) &
    (Injury['longitude'].between(-180, -45))]

# Distinct rows by ID variables
Injury = Injury.drop_duplicates(
    subset=['activity_id', 'vessel_id'], keep='first')
```

```
# Save to external file
if file_export_enabled :
    Injury.reset_index().to_feather(export_data_folder + '/' + 'Injury.feather')
    print(f'Injury {Injury.shape} exported to {export_data_folder}')
else:
    Injury = pd.read_feather(export_data_folder + '/' + 'Injury.feather')
    print(f'Injury {Injury.shape} imported to {export_data_folder}')
```

```
Injury (10367, 14) imported to DataCasualtyAndPollution
```

2.4. VslPoll

In [8]:
```
# Read text file and creating a DataFrame
VslPoll = pd.read_csv(import_data_folder + '/' + 'MisleVslPoll.txt',
                      delimiter='\t',
                      encoding='ISO-8859-1')

# Dataframe variable names from variableNames
VslPoll.columns = variableNames.query('Table == "MisleVslPoll"')['Name'].tolist()

# Variable preselection
VslPoll = VslPoll.drop(['case_id', 'fk_d_vessel', 'vessel_service', 'vessel_type',
                        'vessel_subtype','substance_name', 'substance_class',
                        'substance_subclass', 'substance_type', 'substance_subtype',
                        'discharge_amnt_water', 'discharge_amnt_land', 'discharge_amnt_air',
                        'discharge_amnt_enclosed', 'potential_amnt_total',
                        'potential_amnt_water', 'potential_amnt_land', 'potential_amnt_air',
                        'potential_amnt_enclosed', 'contained_amnt', 'discharge_potential_type',
                        'discharge_situation_type', 'discharge_estimated_land',
                        'discharge_estimated_air', 'discharge_estimated_water',
                        'discharge_estimated_encl', 'potential_case', 'potential_estimated',
                        'contained_estimated', 'unit_of_measure'], axis=1)

# Filter by coords
VslPoll = VslPoll[
    (VslPoll['latitude'].between(15, 70)) &
    (VslPoll['longitude'].between(-180, -45))]

# Distinct rows by ID variables
VslPoll = VslPoll.drop_duplicates(subset=['activity_id', 'vessel_id'], keep='first')

# Save to external file
if file_export_enabled :
    VslPoll.reset_index().to_feather(export_data_folder + '/' + 'VslPoll.feather')
    print(f'VslPoll {VslPoll.shape} exported to {export_data_folder}')
else:
    VslPoll = pd.read_feather(export_data_folder + '/' + 'VslPoll.feather')
    print(f'VslPoll {VslPoll.shape} imported to {export_data_folder}')
```

```
VslPoll (21827, 14) imported to DataCasualtyAndPollution
```

2.5. Activity

In [9]:
```
# Read text file and creating a DataFrame
Activity = pd.read_csv(import_data_folder + '/' + 'MisleActivity.txt',
                       delimiter='\t',
                       encoding='ISO-8859-1')

# Dataframe variable names from variableNames
Activity.columns = variableNames.query('Table == "MisleActivity"')['Name'].tolist()

# incident_dt variable set up
Activity['date'] = pd.to_datetime(Activity['incident_dt'], format="%m/%d/%Y")

# damage_assessment variable set up
Activity['damage_assessment'] = (
    Activity['vessel_property_damage'] +
    Activity['cargo_property_damage'] +
    Activity['facility_property_damage'] +
    Activity['other_property_damage'])

# Variable preselection
Activity = Activity.drop(['case_id', 'incident_dt', 'dept_name', 'activity_type',
                          'activity_status', 'activity_status_subtype',
                          'vessel_property_damage', 'cargo_property_damage',
                          'facility_property_damage', 'other_property_damage'],
                         axis=1)

# Filter by date
Activity = Activity[(Activity['date'].between(pd.to_datetime('2002-01-01', format="%Y-%m-%d"),
                                              pd.to_datetime('2015-12-31', format="%Y-%m-%d")))]

# Distinct rows by ID variables
Activity = Activity.drop_duplicates(subset=['activity_id', 'date'], keep='first')

# Save to external file
if file_export_enabled :
    Activity.reset_index().to_feather(export_data_folder + '/' + 'Activity.feather')
    print(f'Activity {Activity.shape} exported to {export_data_folder}')
else:
    Activity = pd.read_feather(export_data_folder + '/' + 'Activity.feather')
    print(f'Activity {Activity.shape} imported to {export_data_folder}')
```

```
Activity (104476, 4) imported to DataCasualtyAndPollution
```

3. Events: Geographic classification

In [10]:
```python
# This dataframe will be used for the weather data managing
Events = VslEvents.copy()
```

3.1. Region (new variable)

In [11]:
```python
# Function definition for region coords
def assign_region(row):
    latitude = row['latitude']
    longitude = row['longitude']

    if 49 <= latitude <= 70 and -180 <= longitude <= -122:
        return "Alaska"
    elif 49 <= latitude <= 70 and -122 <= longitude <= -45:
        return "Canada"
    elif 15 <= latitude <= 49 and -81.5 <= longitude <= -45:
        return "East Coast"
    elif 15 <= latitude <= 49 and -180 <= longitude <= -100:
        return "West Coast"
    elif 15 <= latitude <= 31 and -100 <= longitude <= -81.5:
        return "Gulf of Mexico"
    elif 31 <= latitude <= 49 and -100 <= longitude <= -81.5:
        return "Mississippi"
    else:
        return "Other Region"

# Function apply to each row of dataframe
Events['region'] = Events.apply(assign_region, axis=1)

# Check new variable counts
print(Events['region'].value_counts())
```

```
region
Gulf of Mexico    36805
East Coast        30162
Mississippi       29011
West Coast        18145
Alaska             6288
Canada               22
Name: count, dtype: int64
```

Region visualization

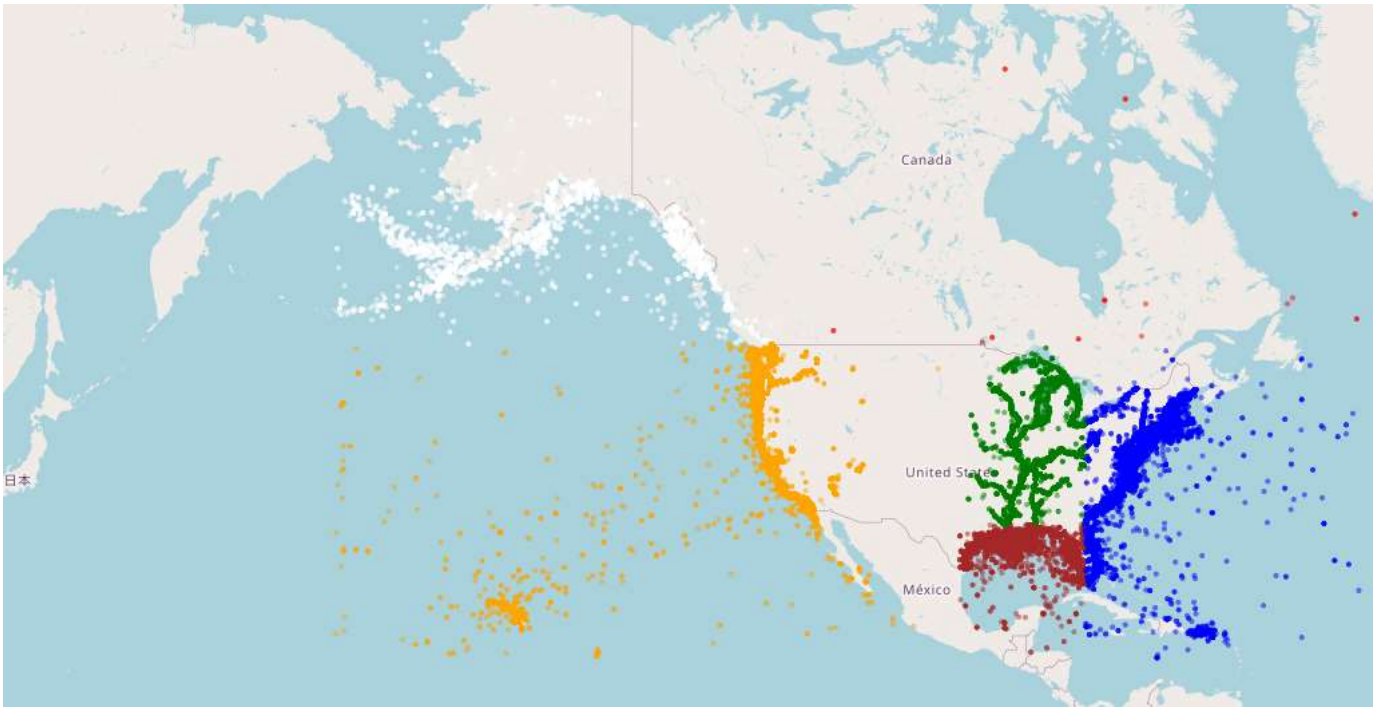
In [12]:
```python
# Region colors set up
colors = {'Alaska': 'white', 'Canada': 'red', 'East Coast': 'blue',
          'West Coast': 'orange', 'Gulf of Mexico': 'brown', 'Mississippi': 'green',
          'Other Region': 'black'}

# Create figure object
fig = go.Figure()

# Aggregate Events points
fig.add_trace(go.Scattermapbox(
    lat=Events['latitude'],
    lon=Events['longitude'],
    mode='markers',
    marker=dict(size=5, color=Events['region'].map(colors), opacity=0.5),
    text=Events.apply(lambda row:f"region:{row['region']}<br>activity_id: {row['activity_id']}", axis=1)))

# Set up map design
fig.update_layout(
    margin ={'l':0,'t':0,'b':0,'r':0},
    mapbox = {'style': "open-street-map", 'center': {'lon': -112, 'lat': 48}, 'zoom': 2})

# Show map
fig.show()
```



3.2. Watertype: river / ocean (new variable)

In [13]:
```
# Based on https://www.matecdev.com/posts/point-in-polygon.html

# Load Shapefile for only EEUU and surroundings (from rnaturalearthdata library)
gdf = gpd.read_file(maps_folder + '/' + 'continental.shp')

# Create a GeoDataFrame from the point DataFrame
geometry = [Point(lon, lat) for lon, lat in zip(Events['longitude'], Events['latitude'])]
gdf_points = gpd.GeoDataFrame(Events, geometry=geometry, crs=gdf.crs)

# Perform spatial join between the GeoDataFrame of points and the GeoDataFrame of polygons
result_sjoin = gpd.tools.sjoin(gdf_points, gdf, how="left", predicate='within')

# The 'index_right' column will contain the indices of the polygons where each point is located
# Create a new variable in your original DataFrame: ocean or river
Events['watertype'] = result_sjoin['index_right'].notnull().astype(int)
Events['watertype'] = np.where(Events['watertype'] == 0, 'ocean', 'river')

# Check new variable counts
Events['watertype'].value_counts()
```

Out[13]:
```
watertype
river    68189
ocean    52244
Name: count, dtype: int64
```

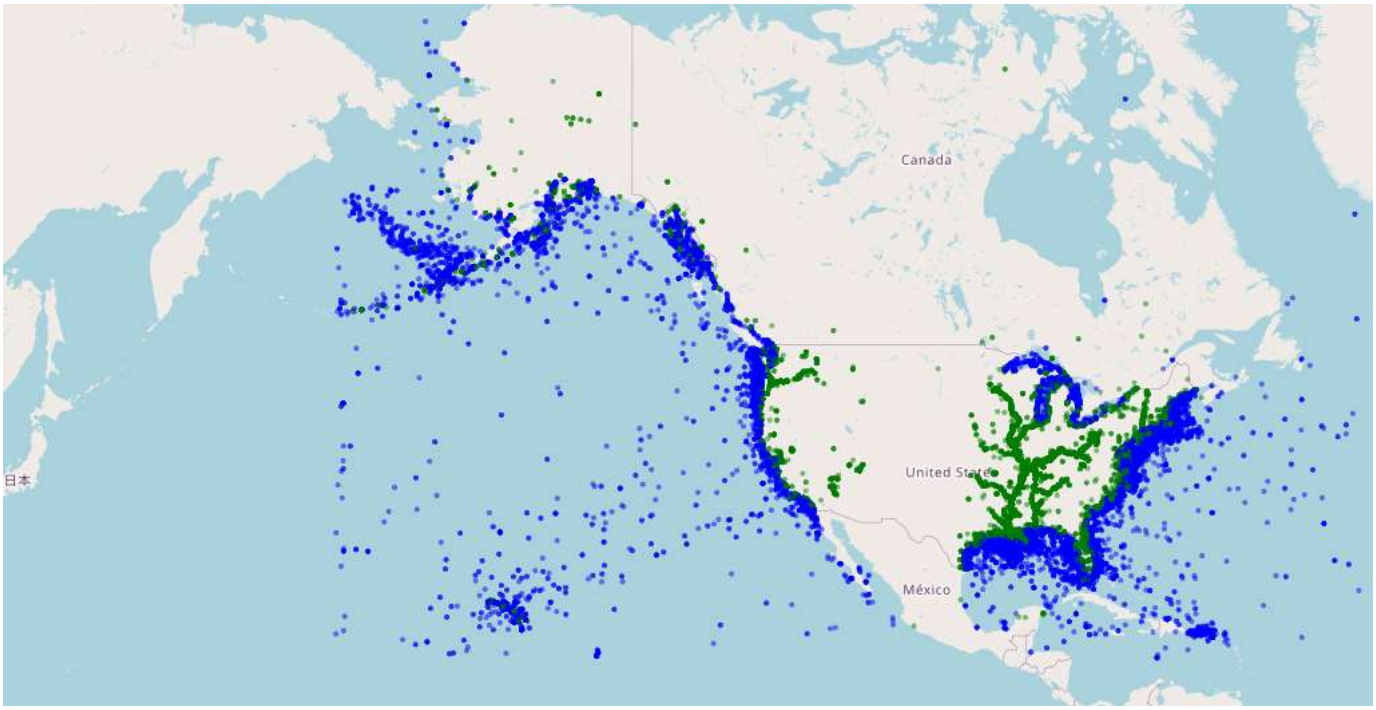
Watertype visualization

In [14]:
```
# Create figure object
fig = go.Figure()

# Aggregate Events points
fig.add_trace(go.Scattermapbox(
    lat=Events['latitude'],
    lon=Events['longitude'],
    mode='markers',
    marker=dict(size=5,
                color=Events['watertype'].map({'ocean': 'blue', 'river': 'green'}),
                opacity=0.5),
    text=Events.apply(lambda row:f"watertype:{row['watertype']}<br>activity_id: {row['activity_id']}", axis=1)))

# Set up map design
fig.update_layout(
    margin ={'l':0,'t':0,'b':0,'r':0},
    mapbox = {'style': "open-street-map", 'center': {'lon': -112, 'lat': 48}, 'zoom': 2})

# Show map
fig.show()
```



3.3. Watertype delimitation

In [15]:
```
# Filter application: only north-american oceans and Mississippi river
Events = Events[(Events['watertype'] != 'river') | (Events['region'] == 'Mississippi')]
Events = Events.dropna(subset=['vessel_id'])

# Print first observations
Events.head()
```

Out[15]:

	index	activity_id	vessel_id	vin	vessel_name	vessel_class	flag_desc	vessel_activity_role_desc	waterway_name	event_type	damage_status	
0	0	4574216	1028411.0	9359052	MATHILDE MAERSK	General Dry Cargo Ship	DENMARK	Involved in a Marine Casualty	GULF OF SANTA CATALINA	Material Failure (Vessels)	Damaged	3
1	1	4574216	1028411.0	9359052	MATHILDE MAERSK	General Dry Cargo Ship	DENMARK	Involved in a Marine Casualty	GULF OF SANTA CATALINA	Vessel Maneuverability	Damaged	3
2	2	4584618	450386.0	1049302	WGN 9713	Barge	UNITED STATES	Involved in a Marine Casualty	TENNESSEE TOMBIGBEE WATERWAY	Grounding	Undamaged	3
3	3	4584618	450386.0	1049302	WGN 9713	Barge	UNITED STATES	Involved in a Marine Casualty	TENNESSEE TOMBIGBEE WATERWAY	Set Adrift	Undamaged	3
4	4	4623838	51354.0	620859	CLARA B	Offshore	UNITED STATES	Involved in a Marine Casualty	GULF DEEP WATER ACCESS	Allision	Damaged	2

3.4. Dataframe export

In [16]:
```
# Save to external file
if file_export_enabled :
    Events.reset_index().to_feather(export_data_folder + '/' + 'Events.feather')
    print(f'Events {Events.shape} exported to {export_data_folder}')
else:
    print('Events {Events.shape} already exported to {export_data_folder}')
```

```
Events {Events.shape} already exported to {export_data_folder}
```