

An Analysis of Canadian Marine Accident Fatality Factors

Name: Rares Finatan

Student Number: 501140875

Supervisor: Uzair Ahmad, Ph.D

Week of December 5, 2022

Table of Contents

Section 1: Abstract	3
Section 2: Literature Review, Data Description, and Methodology	4
2.1 - Defining the Research Question	4
2.2 - Approach of Research	4
2.3 - Literature Analysis.....	4
2.3.1 - Summary of Literatures	4
2.3.2 - Canadian-Specific Literatures.....	5
2.3.3 - International Literatures	11
2.4 - Data Description	12
2.5 - Methodology	17
Section 3: Initial Analysis	19
3.1 - Fundamental Data Analysis	19
3.2: Data Cleaning	23
3.2.1 - Basic Data Cleaning, Primitive Dimensionality Reduction.....	23
3.2.2 - Contextual Data Cleaning.....	25
3.2.3 - Removing Duplicate Attributes and Keeping Integer Encoding	27
3.2.4 - Removing Attributes Unfit for Encoding	28
3.2.5 - Removing Attributes with Low Variance.....	28
3.2.6 - Removing Attributes Irrelevant to Hypothesis.....	29
Section 4: Target Attribute Creation, and Building Train, Validation, Testing Sets	29
4.1 - Target Variable Creation.....	29
4.2 - Data Splitting.....	31
Section 5: Formal Dimensionality Reduction	32
5.1 - Removing Attributes with High Correlations.....	32
5.2 - Dimensionality Reduction by Random Forests Ensemble	34
Section 6: Model Building and Model Comparisons	37

6.1 - Random Forests Ensemble: Model Building and Evaluation	37
6.1.1 - Redefining the Random Forests' Optimal Weighting using Grid Search	40
6.1.2 - Creating a Balanced Random Forests Algorithm.....	45
6.1.3 - Comparison of Random Forest Model Variants	49
6.2 - Weighted Logistic Regression: Model Building and Evaluation	50
6.3 - Naive Bayes Classifier: Model Building and Evaluation.....	55
Section 7: Grouped Model Evaluations	58
Section 8: Conclusions	62
References	64
Github Repository	65

Section 1: Abstract

Marine accidents in Canada's waters affect the lives of Canadian citizens and foreign individuals every year. On an annual basis, the Transportation Safety Board (TSB) of Canada is responsible for maintaining an accurate understanding of all marine incidents via its Marine Safety Information Systems (MARSIS) database. TSB reports that on average between 200-300 marine incidents occur within Canadian territorial waters, of which 16-25% result in serious injuries, and 5-7.5% end in fatalities¹.

Despite modern marine safety standards, accident rates per annum have not decreased significantly since 2010. The research question is to identify the key factors involved in Canadian marine accident fatalities and quantify the respective factors' contributions to serious marine incidents.

Analysis will be conducted on a marine occurrence dataset spanning from 1995 to 2022² using Python. Techniques used in this analysis include essential data cleaning, exploratory analysis coupled with summary statistics, and dimensionality reduction in the form of feature selection following calculations of feature importance. In addition, random forests, logistic regression, and Naive Bayes models will be created to predict the probability of a marine incident resulting in a fatality.

¹ Government of Canada, T. S. B. of C. (2020, July 17). *Marine transportation occurrences in 2020*. Statistical Summary - Transportation Safety Board of Canada. Retrieved September 24, 2022, from <https://www.bst-tsb.gc.ca/eng/stats/marine/2020/ssem-ssmo-2020.html>

² Transportation Safety Board of Canada. (n.d.). *Marine occurrence data from January 1995 to present - occurrence*. Open Government Portal. Retrieved September 24, 2022, from <https://open.canada.ca/data/en/dataset/ad8d1b73-df09-4521-9bdb-61c529328218/resource/1a548829-f7f8-4c2e-a344-a707b13e01c7>

Section 2: Literature Review, Data Description, and Methodology

2.1 - Defining the Research Question

The main topic of research in this study is the analysis of marine accidents occurring in Canadian waters for the years 1995 to 2022. More specifically, the research question is aimed at identifying the key factors involved in Canadian marine accident fatalities and quantifying the respective factors' contributions to serious marine incidents.

A formalized version of the research question reads as follows:

For the years 1995 to 2022, what key factors involved in Canadian marine accidents have quantifiably shown a statistically significant contribution to marine accident fatalities?

2.2 - Approach of Research

All literature articles that are selected for the purpose of this literature review were found using the ProQuest Ebook Central in collaboration with Ryerson University's library, Google Scholar searches, as well as the Transportation Safety Board of Canada's database on marine transportation safety.

2.3 - Literature Analysis

2.3.1 - Summary of Literatures

A large majority of available studies pertain to marine accident investigations conducted in bodies of water not belonging to Canada. Given the scope of the research question being limited to Canadian waters, only investigations originating from the Transportation Safety Board of Canada are directly applicable. Despite this, international or cross-national studies may prove useful in determining research methods and areas of focus not explored by Canadian investigators.

Investigations conducted on Canadian marine transportation safety incidents are typically conducted on a per-occurrence basis³. This means that investigations are conducted on specific maritime incidents, and studies are not often formed to analyze marine accidents on aggregate. Given that the scope of the research question looks to identify summarized variables influencing marine accidents, the literatures under review will also be considered only if they take on a macro-level approach to the research question.

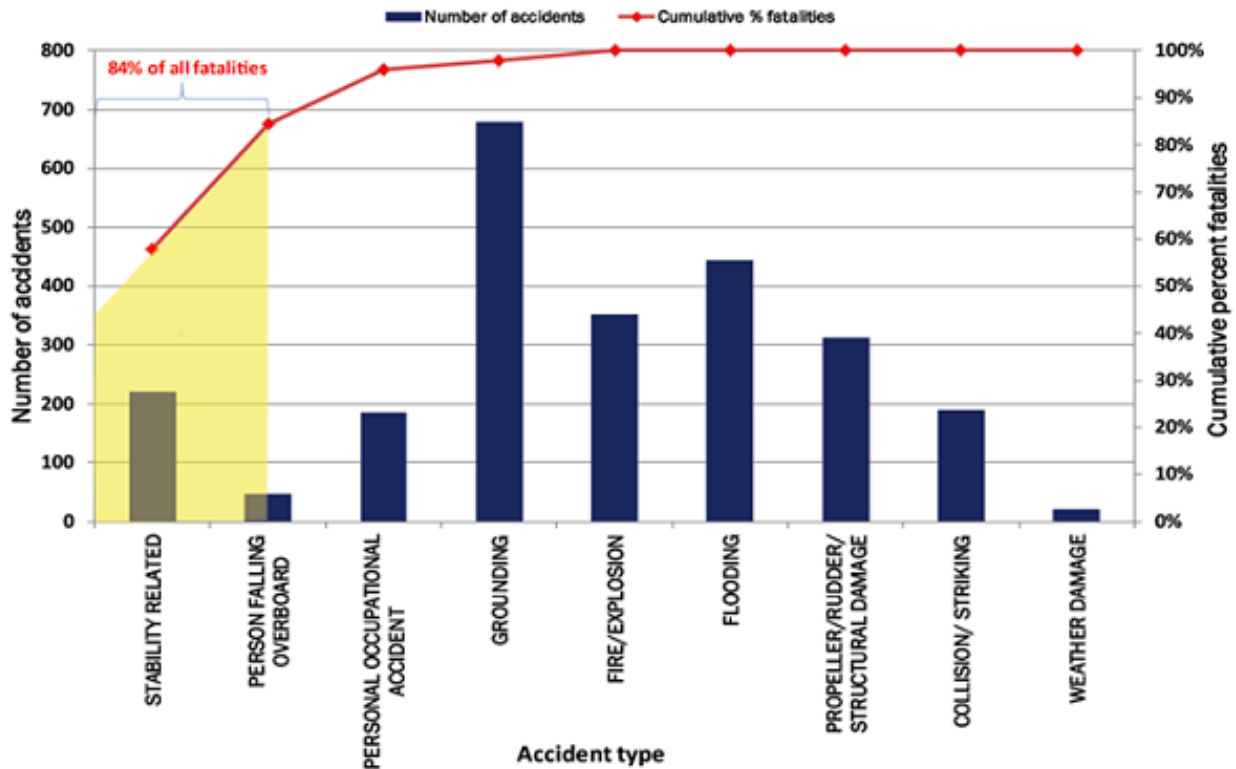
2.3.2 - Canadian-Specific Literatures

1. Transportation Safety Board Investigation M09Z0001: Safety Issues Investigation into Fishing Safety in Canada⁴

This study on marine accidents in Canada is published by the Transportation Safety Board of Canada (TSB), with analysis completed for both public and private sector vessels involved in a marine incident between 1999 and 2010. It is arguably one of the most comprehensive studies of its type, identifying the context behind the impact of commercial vessels and their involvement in marine incident rates, the accident and fatality rates for the respective period, as well as the most pertinent safety issues contributing to marine accident fatalities. The study's results are a compilation of analyses conducted by investigators with expertise across multiple marine disciplines, with a focus on observation and interpretation of 370 investigation reports, 42 safety recommendations, and 100 safety advisory letters published prior to the report release date.

³ Government of Canada, T. S. B. of C. (2019, May 6). *Marine Transportation Safety Investigations and reports*. Transportation Safety Board of Canada. Retrieved October 23, 2022, from <https://www.tsb.gc.ca/eng/rappports-reports/marine/index.html>

⁴ Government of Canada, T. S. B. of C. (2012, August 10). *Marine investigation report M09Z0001*. Marine Investigation Report M09Z0001 - Transportation Safety Board of Canada. Retrieved October 23, 2022, from <https://www.tsb.gc.ca/eng/rappports-reports/marine/etudes-studies/m09z0001/m09z0001.html>



A visual analysis of marine accident types conducted as part of the study.⁵

While the TSB study is highly exhaustive in preparing future safety documentation for marine safety, it has a few limitations, and thus presents several information gaps that can be attenuated in the currently proposed study.

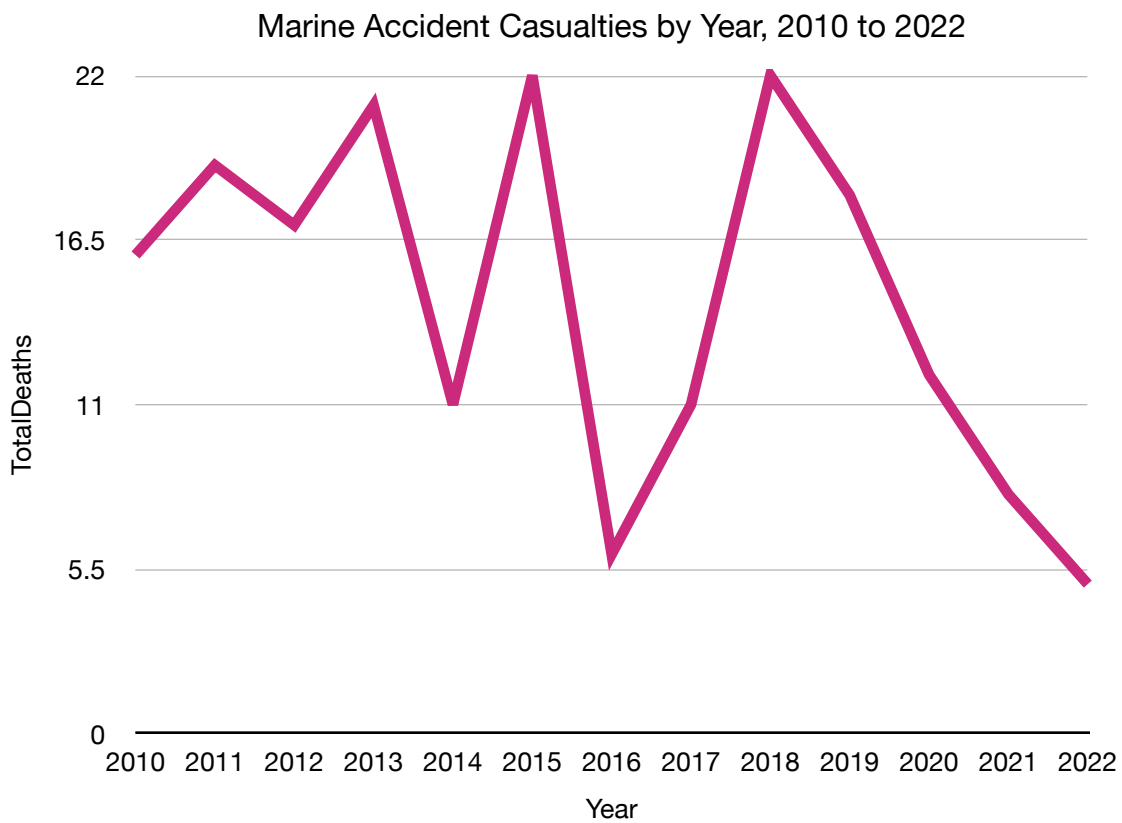
i. The dataset under analysis is strictly constrained to marine vessels operating under the classification of a fishing vessel and does not take into account marine accidents as holistically as possible. More specifically, the investigation does not analyze any marine incidents for the respective time period unless the incident involved a commercial fishing vessel under 24.4 metres in length, or 150 gross tons.

ii. The study does not quantify incident exposure risk from variables that may interact with the 100+ safety actions common to Canadian fishing vessel accidents involved in this

⁵ Transportation Safety Board of Canada. (2010). *Figure 4. Safety Issues Investigation into Fishing Safety in Canada*. Government of Canada. Retrieved October 23, 2022, from <https://www.tsb.gc.ca/eng/rapports-reports/marine/etudes-studies/M09Z0001/images/m09z0001-figure-04.png>.

study. Despite this, there is a qualitative attempt made by the study to consolidate these findings as part of Table 3, Section 7.

iii. The study only contains data from 1999 to 2010, and may suffer from data freshness. Despite updated marine safety standards since the report's release date, holistic marine accident rates per annum have not decreased significantly since 2010. There may be influencing factors not under consideration in the study that have since become sizeable for observation since the report's release date.



Observed marine accident casualties over time in Canada, from 2010 to 2022.⁶

⁶ Transportation Safety Board of Canada. (2019, June 12). *Marine occurrence data from January 1995 to present*. Open Government Portal. Retrieved October 23, 2022, from <https://open.canada.ca/data/en/dataset/ad8d1b73-df09-4521-9bdb-61c529328218>

2. Transportation Safety Board Investigation SM9501: A Safety Study of The Operational Relationship Between Ship Masters/Watch-keeping Officers and Marine Pilots⁷

This study is the second safety issue investigation released by the Transportation Safety Board of Canada, pertaining to the operational relationship between ship masters/watch-keeping officers and marine pilots. The dataset of the study involved 273 marine accident occurrences between February 1981 and May 1992, whereby collisions, groundings, strikings, contacts, and sinking of vessels were reported. Unlike Transportation Safety Board Investigation M09Z0001, there was no self imposed limitation on type, size, or activity of vessel in Canadian waters. Of the 273 incidents considered as part of the study, the research question of the investigation focused on a subset of 200 accidents involving erroneous human factors. This included vessel accident occurrences primarily influenced by misunderstandings, inattention, lack of communication, misjudgement, and miscellaneous human factors.

Of the 200 accident subset wherein human error was involved, statistical analysis was conducted on a manufactured dataset from responses originating from a standardized questionnaire. As part of data collection, groups of respondents (pilots, ship masters, and officers) provided information on the attitudes, behaviours, and interactions among marine personnel involved in the respective accidents.

The study concluded that improved standards of communication, reduction of language barriers, increased monitoring of vessel movements, and greater cooperation amongst crew members were all necessary in order to improve marine safety standards.

⁷ Government of Canada, T. S. B. of C. (1995, January 1). *A SAFETY STUDY OF THE OPERATIONAL RELATIONSHIP BETWEEN SHIP MASTERS/ WATCHKEEPING OFFICERS AND MARINE PILOTS*. Marine Investigation Report SM9501 - Transportation Safety Board of Canada. Retrieved October 23, 2022, from <https://www.tsb.gc.ca/eng/rapports-reports/marine/etudes-studies/SM9501/SM9501.html>

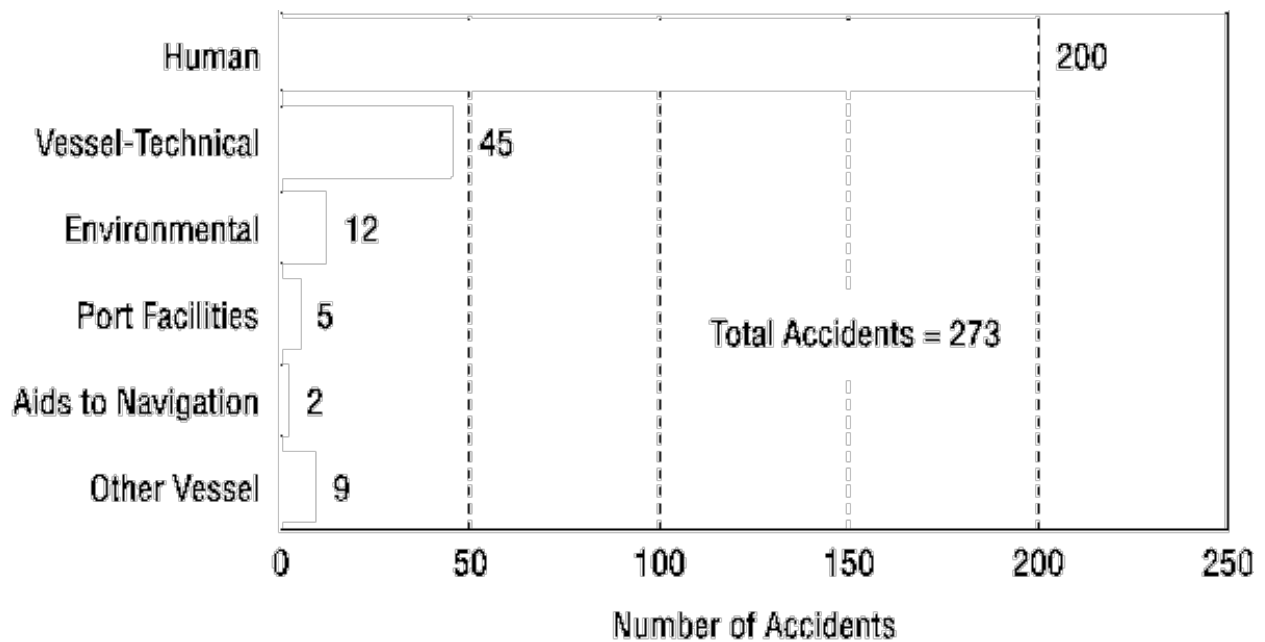


Figure 1, Section 2.3.2, showcasing the propensity of human error in marine accidents between 1981 and 1992.⁸

The study presents multiple findings on the importance of minimizing human error within marine safety procedures that had not been explored thoroughly prior to the report's release date. While it investigates the scope of marine safety on a similar basis when compared to Transportation Safety Board Investigation M09Z0001, SM9501 limits its overall ability to derive insights due to several research limitations:

i. The study presents an inherent bias in the types of vessels under investigation. The dataset primarily focuses on marine accident occurrences where 87% of the 273 occurrences were involving vessels greater than 5,000 gross registered tons. This impacts the scope of data that could have been captured in occurrences involving small registered vessels, which could possibly experience similar human error scenarios. This presents an opportunity for the current study proposal which aims to consider all vessel classes, weights, and sizes to provide an overall understanding of human and non-human factors affecting marine accident fatality rates.

⁸ Transportation Safety Board of Canada. (1995). *Figure 1*. Transportation Safety Board of Canada. Government of Canada. Retrieved October 23, 2022, from <https://www.tsb.gc.ca/eng/rapports-reports/marine/etudes-studies/SM9501/images/ems9501a.gif>.

ii. The contributing factors under study aimed to assess the impact of human error on marine accidents and fatalities. However, of the attributes under consideration, no direct interaction calculation was attempted between human and non-human factors. It is possible that non-human factors, such as technical failures of the vessels, environmental conditions, port facilities conditions, and interference from other vessels could impact the contributing human factors under study.

iii. As briefly mentioned in the critical review of Transportation Safety Board Investigation M09Z0001, SM9501 also suffers from data freshness given the dataset was collected between 1981 and 1992. The survey data gathered may also be subject to survey question bias given that “[...]each question began with the prefix “In my experience....”[...]”, and did not offer any other form of questionnaire data input.

The dataset of the proposed study does not need to contend with an issue of bias given that it is factual data of marine accident occurrences simply logged monthly. However, the data under proposal may be susceptible to accident occurrence validation given its high-level of data freshness⁹.

⁹ Transportation Safety Board of Canada. (2019, June 12). *Marine occurrence data from January 1995 to present*. Open Government Portal. Retrieved October 23, 2022, from <https://open.canada.ca/data/en/dataset/ad8d1b73-df09-4521-9bdb-61c529328218>

2.3.3 - International Literatures

3. An Analysis of Factors Affecting the Severity of Marine Accidents¹⁰

This study comprises of a statistical analysis of 1,207 marine accidents occurring across the globe from 2010 to 2019. The data collected for this study are from 7 marine safety boards, inclusive of the Transportation Safety Board of Canada, the Marine Accident Investigation Branch of the United Kingdom, the Australian Transport Safety Bureau, the United States Office of Marine Safety, the German Federal Bureau of Maritime Casualty Investigation, the China Maritime Safety Administration, and the Japan Transport Safety Board. Given various severities of marine accident categories, Wang et al. explore the influencing factors of the severity of marine accidents using an ordered logistic regression model.

The results of the study are able to provide a probability of the positive associative factors resulting in a higher marine accident severity. In addition, the same results are individually documented for types of vessels, as well as seven accident categories: collision, grounding, fire/explosion, contact, sinking, equipment failure, and others. Relativity of accident severity ratios are provided for different influencing factors as part of the study, inclusive of accident type, human element, ship type, ship condition, and environmental interference. The study aims to address some of the limitations of the aforementioned Canadian literature by including human, technical, and environmental factors as part of the interactions leading up to marine accident occurrences.

¹⁰ Wang, H., Liu, Z., Wang, X., Graham, T., & Wang, J. (2021). An analysis of factors affecting the severity of marine accidents. *Reliability Engineering & System Safety*, 210, 107513. doi:10.1016/j.ress.2021.107513

2.4 - Data Description

The dataset under study is titled “*Marine occurrence data from January 1995 to present*” and is published in CSV format by the Government of Canada in cooperation with the Transportation Safety Board of Canada. The data is directly sourced from the Marine Safety Information Systems (MARSIS) operated by the TSB. Notably, the data set is refreshed on or soon after the 15th of every month. As such, there are some small changes that may occur between the time of writing this data description and the final version of the data set used in model production. The full data set ranges from January 1995 to the present day, up until the latest cycle of 15 days occurring at the time of writing. The full data set operates under an Open Government Licence by the Government of Canada and can be found at this URL:

https://www.tsb.gc.ca/includes/stats/csv/Marine/MARSISdb_MDOTW_VW_OCCURRENCE_PUBLIC.csv

For the respective marine occurrences under study, there are also records on the relevant vessels, navigation equipment, lifesaving appliances equipment, recording equipment, and injuries descriptions. These are all distributed as separate CSV files, but will not be under study as part of the core hypothesis analysis. There may be additional academic links made to the CSVs for explanatory or descriptive purposes, but not statistical or quantifiable analysis.

The raw dataset contains 83,797 entries across 160 attributes. For the analysis under study, not all 160 attributes will be considered. To view all 160 attributes, a data dictionary has been compiled by the TSB and is accessible [here](#).

To understand the impact of attributes contributing to marine accident fatalities, a subset of the data must be taken, whereby only severe accidents resulting in fatalities have occurrence records. The dataset by default classifies occurrences by International Maritime Organization class levels, as well as TSB investigation classification levels of the occurrence. These two classifications differ in the way they numerically attribute fatalities to the

occurrences - subsequently, it is most prudent to simply look at occurrences which have non-zero fatality records.

```
import pandas as pd
```

```
marsis_raw =  
pd.read_csv("MARSISdb_MD0TW_VW_OCCURRENCE_PUBLIC.csv")  
marsis_deadly = marsis_raw[marsis_raw["TotalDeaths"] > 0]
```

Of interest, there are 3,235 records of fatalities. These records ought to be further subdivided by the severity of the accident leading to the fatality occurrence. Not all fatalities are the result of extreme circumstances, and observing the breakout of accident severity may give an indication of the distribution of accident severity respective to fatality occurrences.

For the purposes of enabling this study to be universally accessible, accident severity is best subcategorized under the International Maritime Organization's standards¹¹. These include three levels of severity, identified as the following:

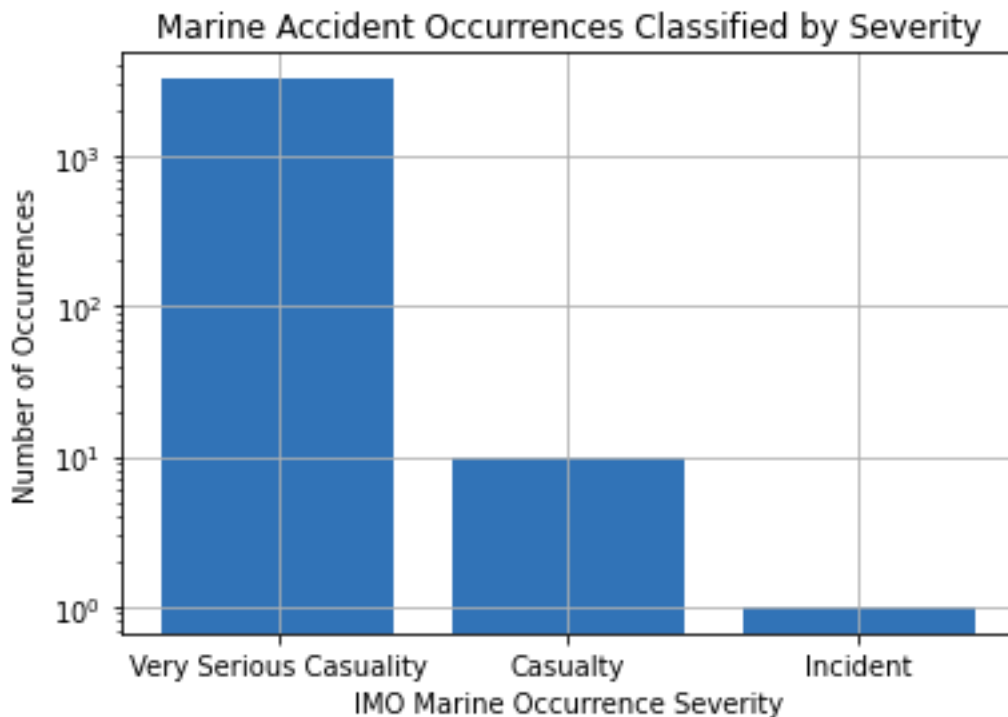
- **Level 1 - Very Serious Marine Casualty:** A marine incident which involves a loss of life, severe pollution, or a total loss of the vessel involved.
- **Level 2 - Marine Casualty:** An occurrence not classified as a very serious casualty, typically involving a fire, explosion, collision, grounding, contact, heavy weather damage, ice damage, hull cracking, or suspected hull defects, resulting in:
 - immobilization of main engines, extensive accommodation damage, severe structural damage, such as penetration of the hull under water, etc., rendering the ship unfit to proceed

¹¹ International Maritime Organization. (2014, November 18). *CASUALTY-RELATED MATTERS* REPORTS ON MARINE CASUALTIES AND INCIDENTS Revised harmonized reporting procedures – Reports required under SOLAS regulations I/21 and XI-1/6, and MARPOL, articles 8 and 12*. International Maritime Organization. Retrieved October 23, 2022, from <https://wwwcdn.imo.org/localresources/en/OurWork/MSAS/Documents/MSC-MEPC3/MSC-MEPC.3-Circ.4%20Rev%201%20%20Revised%20harmonized%20reporting%20procedures%20-%20Reports%20required%20under%20SOLAS%20regulations%20I21.pdf>

- pollution (regardless of quantity)
- a breakdown necessitating towage or shore assistance
- **Level 3 - Marine Incident:** Includes occurrences not classified as very serious or serious casualties. Includes records in which hazardous occurrences or “near misses” took place, and if no corrective action were taken, would have endangered the safety of the vessel, its occupants, or any other person or the environment.

```
names = ["Very Serious Casualty", "Casualty", "Incident"]
values = [
    len(marsis_deadly[marsis_deadly["ImoClassLevelID"] == 1]),
    len(marsis_deadly[marsis_deadly["ImoClassLevelID"] == 2]),
    len(marsis_deadly[marsis_deadly["ImoClassLevelID"] == 3])
]
```

```
plt.bar(names, values)
plt.yscale("log", base=10)
plt.xlabel("IMO Marine Occurrence Severity")
plt.ylabel("Number of Occurrences")
plt.title("Marine Accident Occurrences Classified by Severity")
plt.grid(True)
plt.show()
```



In addition to analyzing the severity classification of each individual fatality, the cause of the accident or incident will be an important variable in feature engineering. Each accident / incident type is provided in the dataset by a presumed attribute named `AccIncTypeDisplayEng`, containing more granular classifications of each incident under study. To understand how many deaths are attributed to marine occurrence severity and their respective impact on total death counts, the following grouping is made to better illustrate the data:

Accident/Incident Type	Severity Class	Total Deaths
ABANDONED	1	131
BOTTOM CONTACT	1	2
CAPSIZES	1	754
	2	2
CARGO SHIFT/CARGO LOSS - Cargo shifted	1	2
COLLISION - Struck by vessel	1	4
COLLISION - With another vessel or other floating object	1	362
EXPLOSION	1	88
	3	1
FIRE	1	165
	2	1
GROUNDING - Not under power (includes drifting) (non-intentional)	1	123
GROUNDING - Under power (non-intentional)	1	79
MISSING (the ship is)	1	239
PERSON SERIOUSLY INJURED OR KILLED - Boarding, being on board, falling overboard from the ship	1	265
	2	6
PERSON SERIOUSLY INJURED OR KILLED - In contact with any part of the ship or its contents	1	1517
RISK OF SINKING	1	2
SANK - Flooding	1	163
SANK - Founders (taking on water above the waterline)	1	1265
	2	1
STRIKING - Allision with a fixed object (striking - includes berthed/docked vessels)	1	40
SUSTAINS DAMAGE RENDER UNSEAWORTHY/UNFIT FOR PURPOSE - Unfit for purpose - ice, weather, etc.	1	14

One assumption that needed also to be tested early on for our hypothesis was the correlation found between injuries sustained in an occurrence and the total deaths in an incident depending on the severity class of the incident. A correlation matrix was created to primarily analyze the directional relationship between total deaths and several attributes commonly occurring in highly severe classes of incidents.

The correlation matrix below analyses the following attributes commonly seen in highly severe classes of incidents:

- **TotalMinorInjuries:** The total number of persons who received minor injuries as a result of the occurrence.
- **TotalSeriousInjuries:** The total number of persons seriously injured as a result of the occurrence.
- **TotalMissingIndividuals:** The total number of persons who are missing as a result of the occurrence.
- **TotalPeopleInTheWater:** The total number of persons in the water as a result of the occurrence.

	TotalDeaths	TotalMinorInjuries	TotalSeriousInjuries	TotalMissingIndividuals	TotalPeopleInTheWater
TotalDeaths	1.00	0.05	0.01	0.05	0.05
TotalMinorInjuries	0.05	1.00	0.09	0.03	-0.00
TotalSeriousInjuries	0.01	0.09	1.00	-0.02	0.07
TotalMissingIndividuals	0.05	0.03	-0.02	1.00	-0.01
TotalPeopleInTheWater	0.05	-0.00	0.07	-0.01	1.00

Given the per occurrence correlation between total deaths and the aforementioned attributes, there appears to be no correlation (or an extremely weak positive correlation). As there is little directionality in the correlation matrix to indicate a relationship between the suspected attributes, additional exploratory analysis will need to be conducted during feature engineering. There is also additional intra-attribute analysis that needs to be conducted in order to determine deeper relationships between suspected (and not already considered) variables.

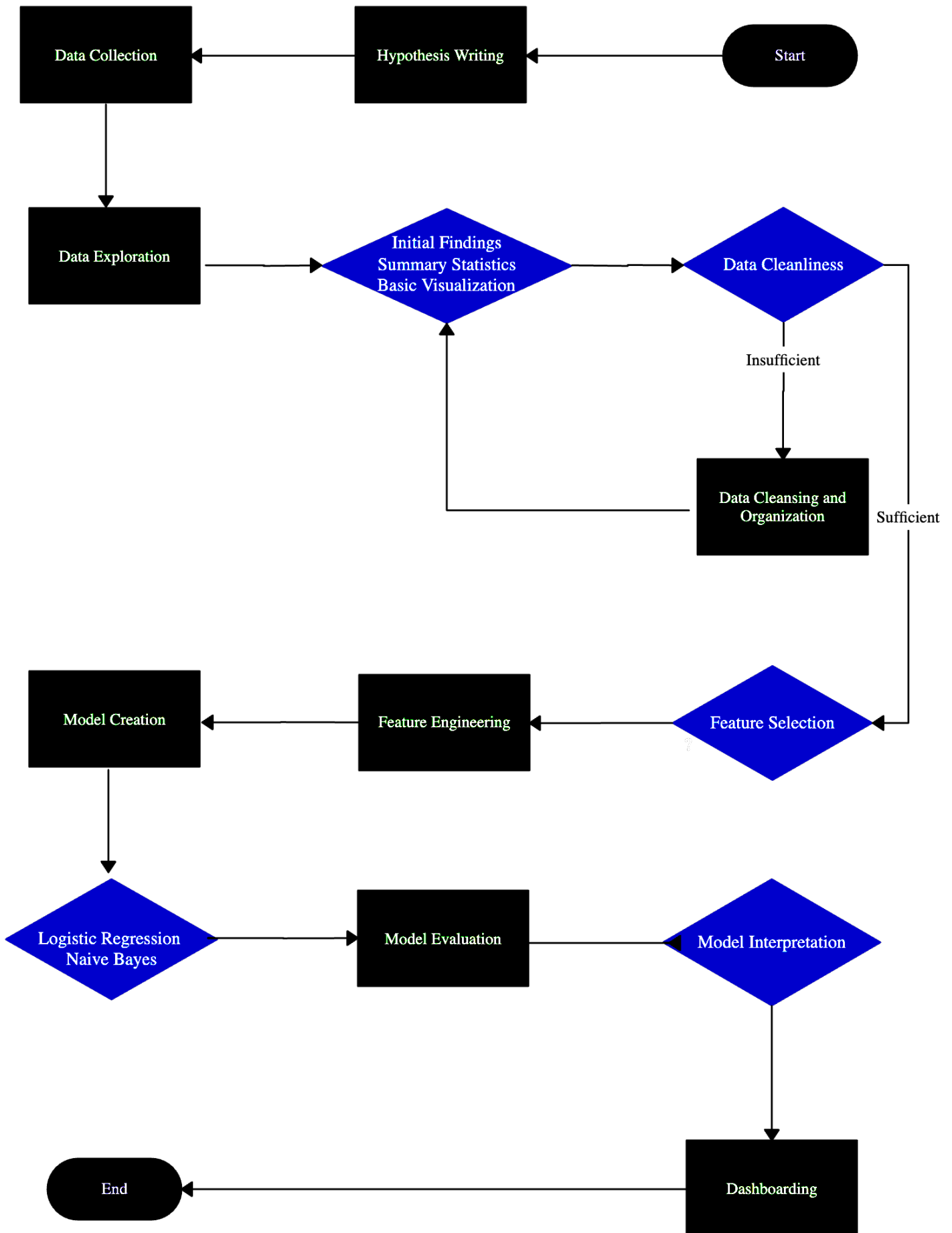
Lastly, some summary statistics on fatality occurrences provide us with an understanding of some of the dataset's aforementioned attributes and the range of values that are appropriate for further analysis and interpretation.

	TotalDeaths	TotalMinorInjuries	TotalSeriousInjuries	TotalMissingIndividuals	TotalPeopleInTheWater
mean	1.46	0.19	0.10	0.16	0.11
std	2.90	0.78	0.52	1.11	0.83
min	1.00	0.00	0.00	0.00	0.00
25%	1.00	0.00	0.00	0.00	0.00
50%	1.00	0.00	0.00	0.00	0.00
75%	1.00	0.00	0.00	0.00	0.00
max	84.00	12.00	8.00	28.00	21.00

2.5 - Methodology

As part of the iterative process of this study, the illustration seen below will act as a primary flow of work to be completed given the established approach thus far.

1. Hypothesis writing and initial problem framing
2. Data collection and data imports
3. Data exploration
 - 3.1. Initial findings, summary statistics
 - 3.2 Data cleanliness and data logic
4. Feature selection
5. Feature engineering
6. Model creation
 - 6.1. Random Forests
 - 6.2 Logistic regression
 - 6.2. Naive Bayes
7. Model evaluation
 - 7.1. Model interpretation
8. Conclusions and final presentation



Section 3: Initial Analysis

3.1 - Fundamental Data Analysis

Before beginning any elementary data analysis, recall the datasets under study:

- [MARSISdb MDOTW VW OCCURRENCE PUBLIC](#) - Data of marine occurrences in Canadian waters from January 1995 to present
- [MARSISdb MDOTW VW OCCURRENCE VESSEL PUBLIC](#) - Vessel-specific data involving marine occurrences in Canadian waters from January 1995 to present

The two datasets are saved locally in the project's working directory from the provided URLs, in .CSV format. Analysis begins with initial data imports of the CSVs, stored as pandas data frames in two respective variables.

```
#initial data import
occurrence_raw = 
pd.read_csv('MARSISdb_MDOTW_VW_OCCURRENCE_PUBLIC.csv')
vessel_raw = 
pd.read_csv('MARSISdb_MDOTW_VW_OCCURRENCE_VESSEL_PUBLIC.csv')
```

For the purposes of this study, both occurrence-specific attributes and vessel-specific attributes are taken under consideration in order to analyze their respective impact on fatality occurrences. The two datasets share the same occurrence primary key, `OccID`. In order to have both datasets' attributes in a single data frame, they are merged using the following command:

```
#merge the raw data sets
marsis_raw = pd.merge(
    occurrence_raw,
    vessel_raw,
    how = 'inner',
    left_on = 'OccID',
    right_on = 'OccID')
```

With the merged data frame, one can begin inspecting the structure of the data and its respective attributes. This step is crucial in understanding the amount of rows, columns, and types of attributes comprising each record.

```
#view structure of the dataset  
marsis_raw.info()
```

```
Out[ ]:  
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 137368 entries, 0 to 137367  
Columns: 194 entries, OccID to ActivityCategoryDisplayFre  
dtypes: float64(68), int64(12), object(114)
```

```
#view attribute data types  
marsis_raw.dtypes
```

```
Out[ ]:  
OccID                int64  
OccNo                object  
OccClassID          int64  
OccClassDisplayEng  object  
OccClassDisplayFre  object  
VesselPhaseDisplayFre  object  
Speed_Knots         float64  
ActivityCategoryID   float64  
ActivityCategoryDisplayEng  object  
ActivityCategoryDisplayFre  object  
Length: 194, dtype: object
```

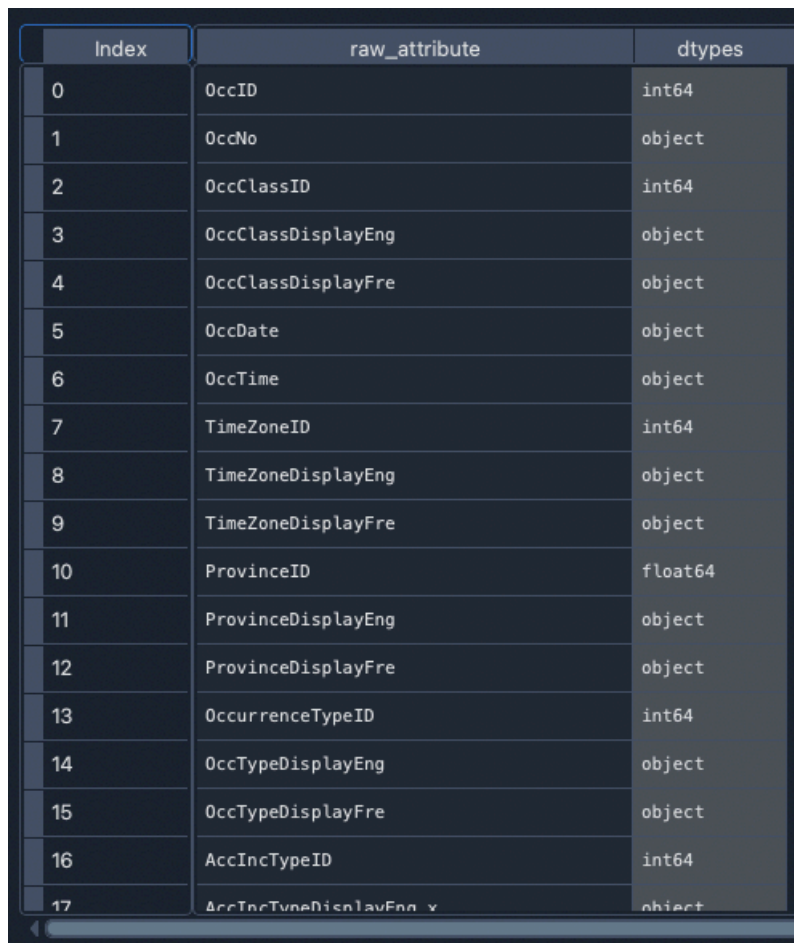
Glimpsing at the combined data frame using pandas also provides a view of some basic descriptive statistics for each attribute.

```
#view structure of combined data frame using pandas  
pd.DataFrame.describe(marsis_raw)
```

	OccID	OccClassID	...	Speed_Knots	ActivityCategoryID
count	137368.000000	137368.000000	...	2728.000000	43642.000000
mean	20354.798199	2.604923	...	6.355374	3.411461
std	12731.055606	2.773323	...	5.635783	2.062087
min	1.000000	1.000000	...	0.000000	1.000000
25%	10146.000000	1.000000	...	1.700000	2.000000
50%	18752.000000	1.000000	...	5.900000	3.000000
75%	28796.000000	4.000000	...	10.000000	6.000000
max	50402.000000	18.000000	...	50.000000	7.000000

For the extracted data types, one can store them into a data frame for further analysis or manipulation. The data types of the columns are stored in a data frame, pointing to a variable named `marsis_data_types`.

```
#create a dataframe of the datatypes in the dataset
marsis_data_types = marsis_raw.dtypes.to_frame('dtypes').reset_index()
marsis_data_types.rename(columns = {'index':'raw_attribute'}, inplace
= True)
```



Index	raw_attribute	dtypes
0	OccID	int64
1	OccNo	object
2	OccClassID	int64
3	OccClassDisplayEng	object
4	OccClassDisplayFre	object
5	OccDate	object
6	OccTime	object
7	TimeZoneID	int64
8	TimeZoneDisplayEng	object
9	TimeZoneDisplayFre	object
10	ProvinceID	float64
11	ProvinceDisplayEng	object
12	ProvinceDisplayFre	object
13	OccurrenceTypeID	int64
14	OccTypeDisplayEng	object
15	OccTypeDisplayFre	object
16	AccIncTypeID	int64
17	AccIncTypeDisplayEng	object

The output data frame containing pandas-identified data types.

With the pandas-identified data types in the merged dataset, one can compare the attributes' data types with the ones provided by the author of the dataset via a data dictionary. The data dictionary for this study is found [here](#), and imported into the IDE as `MARSISdb-dd-processed.csv`, and consequently stored in a new variable `marsis_dd`. The data dictionary

is then merged with the previously created `marsis_data_types` data frame in order to view the pandas-identified data types and the author-identified data types side-by-side. This side-by-side comparison data frame is stored in a new variable, `data_type_comparison`.

```
#compare identified data types to provided data dictionary
marsis_dd = pd.read_csv("MARSISdb-dd-processed.csv")
marsis_dd = marsis_dd[marsis_dd.table_name == 'MDOTW_VW_OCCURRENCE_PUBLIC']
data_type_comparison = pd.merge(
    marsis_data_types,
    marsis_dd,
    how = "inner",
    left_on = "raw_attribute",
    right_on = "column_name"
)
```

raw_attribute	dtypes	table_name	column_name	full_column_name	descriptions	data_type
OccID	int64	MDOTW_VW_0...	OccID	Occurrence ident...	The occurre...	int
OccNo	object	MDOTW_VW_0...	OccNo	Occurrence number	The TSB occ...	varchar
OccClassID	int64	MDOTW_VW_0...	OccClassID	Occurrence class...	A system-ge...	int
OccClassDisp...	object	MDOTW_VW_0...	OccClassDisp...	Occurrence class...	The TSB inv...	varchar
OccClassDisp...	object	MDOTW_VW_0...	OccClassDisp...	Occurrence class...	The TSB inv...	varchar
OccDate	object	MDOTW_VW_0...	OccDate	Occurrence date	The date on...	date
OccTime	object	MDOTW_VW_0...	OccTime	Occurrence time	The time at...	time
TimeZoneID	int64	MDOTW_VW_0...	TimeZoneID	Time zone identi...	A system-ge...	int
TimeZoneDisp...	object	MDOTW_VW_0...	TimeZoneDisp...	Time zone (English)	The time zo...	varchar
TimeZoneDisp...	object	MDOTW_VW_0...	TimeZoneDisp...	Time zone (French)	The time zo...	varchar
ProvinceID	float64	MDOTW_VW_0...	ProvinceID	Province identif...	A system-ge...	int

A glimpse of the first rows of the pandas-identified data types and the authored data dictionary data types.

3.2: Data Cleaning

3.2.1 - Basic Data Cleaning, Primitive Dimensionality Reduction

As part of elementary data cleaning, some inconsistencies in the data set need to be addressed before formal dimensionality reduction. Given that many data analysis tools are unable to accept missing data when training models, elimination of missing data records or columns may be necessary, despite the loss of information incurred. Removing records and columns that are completely empty will not impact the inferential capacity of a model given that no data is present in an attribute.

```
#remove attributes with no values
marsis_no_nas = marsis_raw.copy(deep=True)
marsis_no_nas.dropna(how='all', axis='columns', inplace=True)
```

With this command, the data set under study is reduced from 194 dimensions, to 190. Four attributes had no data records at all.

```
marsis_raw          DataFrame          (137368, 194)
marsis_no_nas       DataFrame          (137368, 190)
```

Given that the data set is published by the Government of Canada and its Transformation Safety Board (TSB), the attributes that are non-numerical are duplicated in both English and French. For the purposes of analyzing the attributes and their record row values, only one language is necessary. Continue cleaning the data set by dropping all French attributes, denoted by a text string ending in 'DisplayFre'.

```
#some columns are duplicated in English and French
#only require the English attribute variants, remove the French
marsis_eng_only = marsis_no_nas.copy(deep=True)
marsis_eng_only.drop(marsis_eng_only.filter(regex='DisplayFre').columns, axis=1, inplace=True)
```

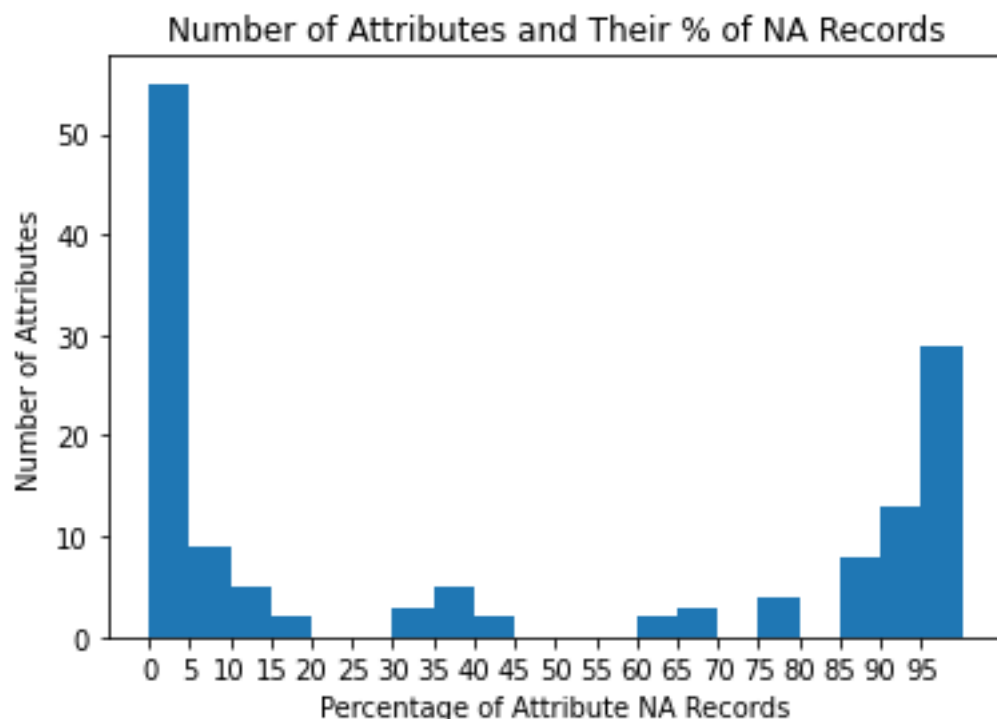
By removing French duplicated attributes, the data set under study is further reduced from 190 dimensions to 140.

In addition to removing attributes which have 100% NA records, one ought to also reduce the amount of dimensions that have NAs that exceed a particular proportionality relative to the overall attribute's records. Determine which attributes have more than 80% NA records, add them to a list, and remove all attributes which comprise of more than 80% NA records.

```
#determine each column's percentage of NA records
nas_percentage = (marsis_eng_only.isna().mean().round(2) *
100).to_frame(name = 'percentage')
#determine the columns with high percentage of NAs, more specifically
greater than 80% NAs
nas_80_percent = nas_percentage.loc[nas_percentage['percentage'] > 80]
```

Next, a visual check is conducted for the anticipated inflection point for NA percentages. One ought to verify if 80% NA records is an appropriate threshold for attribute elimination.

```
#visualize distribution of NA percentages
import matplotlib.pyplot as plt
import numpy as np
plt.hist(nas_percentage['percentage'], bins = 20)
plt.xticks(np.arange(0, max(nas_percentage['percentage']), 5))
plt.title('Number of Attributes and Their % of NA Records')
plt.xlabel('Percentage of Attribute NA Records')
plt.ylabel('Number of Attributes')
plt.show()
```



Judging from the visualization, 80% NAs seems to be a good threshold to remove attributes. There are too few attributes in between 85% and 20% to account for significant dimensionality reduction. Next, a list of attributes is created that matches the 80% NAs condition, and after, the attributes contained in the list are dropped from the overall data set.

```
#create a list of the attributes that have higher than 80% NA records
nas_80_percent.reset_index(inplace=True)
nas_80_percent_list = list(nas_80_percent['index'])

#remove the attributes with more than 80% NA records
marsis_no_nas_80 = marsis_eng_only.drop(columns = nas_80_percent_list)
```

By removing the attributes with more than 80% NA records, the data sets dimensions have been further reduced from 140 to 90.

Continuing with basic dimensionality reduction, a separate source of NAs persists in the processed data set so far, whereby NAs are masked as NULL text strings. All columns which are completely containing NAs masked as NULL text strings are removed.

```
#there are also columns that have NA values masked as NULL strings
#identify the columns with all NULL string records
marsis_nulls = marsis_no_nas_80.loc[:, ((marsis_no_nas_80 ==
'Null').any())]
marsis_nulls = list(marsis_nulls.columns)

#remove the columns with all NULL string records
marsis_basic_cleaned = marsis_no_nas_80.drop(columns = marsis_nulls)
```

With this command, only a single attribute was dropped from the data set, to a new running total of 89 attributes.

3.2.2 - Contextual Data Cleaning

Up until this point, the raw data and its respective iterations of processing had been saved to new variables to account for any retroactive errors or overzealous data cleansing. In addition, the data cleansing techniques employed in section 2.1 were wide-sweeping and general in nature; they are techniques that likely could be deployed to any dataset. The next step is to contextualize data munging relative to the study's hypothesis - identifying

deterministic factors of marine occurrences resulting in fatalities in Canadian waters from 1975 to the present day. From herein, cleansing of the dataset will occur iteratively and have its results stored in a data frame variable named `marsis_processed`.

```
#given the context of the study, the area of focus is occurrences
resulting in deaths
marsis_processed = marsis_basic_cleaned.copy(deep=True)
```

The processed dataset contains many records which are seemingly duplicates. Particularly when analyzing the attributes `OccID` (the unique ID assigned to an occurrence) and `OccNo` (the unique case file number assigned to an occurrence), one can see that multiple `OccID` and `OccNo` records refer to the same event. This is because the Government of Canada's marine authorities are required to log individual phases of an occurrence, while maintaining a record of which occurrences had multiple phases. The recommended cleaning technique is to keep records with the most recent date-time stamps as they are rows containing the final outcome of each marine occurrence.

```
#to identify the unique cases of deadly occurrences, duplicate OccIDs
must be dropped, and only the most recent is kept
#the most recent OccID is kept because it houses the final
investigative data leading to the occurrence's result
```

```
#all OccDate instances have a 12:00:00 timestamp applied to them,
whether actual or not
#strip timestamp from OccDate attribute records
marsis_processed['OccDate'] =
marsis_processed['OccDate'].str.replace(' 12:00:00 AM', '')
```

```
#some OccTime instances have NA values; assume 00:00:00 timestamp
given lack of information
marsis_processed['OccTime'] =
marsis_processed['OccTime'].fillna('00:00:00')
```

```
#concatenate date and time attributes
marsis_processed['OccDateTime'] = marsis_processed['OccDate'] + ' ' +
marsis_processed['OccTime']
```

```
#ensure date attributes are of dtype 'datetime'
marsis_processed['OccDateTime'] =
pd.to_datetime(marsis_processed['OccDateTime'])
```

```
#drop duplicate OccID instances, but keep latest OccID instance
marsis_processed_unique = 
marsis_processed.sort_values('OccDateTime').drop_duplicates('OccID',
keep = 'last')
```

Index	onTypeDispl	HullMaterialID	MaterialDisplay	YearBuilt	icTypeDisplayE	VesselPhaseID	ielPhaseDisplay	ctivityCategory	yCategoryDispl	OccDateTime
137367	OPELLER	3	STEEL	1955	PERSON SERIO...	1	BERTHED/ DOCKED	nan	nan	1975-01-01 12:10:00
137366	OPELLER	3	STEEL	1968	PERSON SERIO...	8	UNDERWAY - unknown	nan	nan	1975-01-03 08:01:00
137365	P. OPELLER	3	STEEL	1973	STRIKING - A...	8	UNDERWAY - unknown	nan	nan	1975-01-04 14:34:00
137356	OPELLER	3	STEEL	1964	PERSON SERIO...	8	UNDERWAY - unknown	nan	nan	1975-01-08 15:45:00
137358	OPELLER	3	STEEL	1952	FIRE	19	UNKNOWN	nan	nan	1975-01-08 19:00:00
137357	OPELLER	3	STEEL	1965	GROUNDING - ...	6	UNDERWAY - moving ahead	nan	nan	1975-01-08 20:40:00
137353	OPELLER	3	STEEL	1972	GROUNDING - ...	6	UNDERWAY - moving ahead	nan	nan	1975-01-09 12:10:00
137355	OPELLER	2	WOOD	1957	FIRE	1	BERTHED/ DOCKED	nan	nan	1975-01-09 18:00:00
137354	OPELLER	2	WOOD	1974	GROUNDING - ...	2	ANCHORED	nan	nan	1975-01-09 18:15:00
137350	OPELLER	2	WOOD	1973	COLLISION - ...	19	UNKNOWN	nan	nan	1975-01-13 11:15:00

A glimpse into a few rows of the dataset showing unique occurrences, with only the most recent date-time stamp intact for each occurrence.

3.2.3 - Removing Duplicate Attributes and Keeping Integer Encoding

Record rows within the dataset are now unique based off of the attribute `OccID`. The processed dataset is now comprised of 45,059 records and 90 attributes, which is still containing many attributes undesirable for model building and evaluation. Within the `marsis_processed` data frame, there are many duplicate attributes. These duplicate attributes are shown in the dataset firstly as an encoded value (using TSB nomenclature), followed by a textual representation of the encoded value. For model creation, one ought to only keep encoded values - any other attribute classes would need to be encoded separately using one-hot encoding, or another preferred method.

```
#remove attributes that have an equivalent attribute, but keep the
attribute that has integer encoding
```

```
#OccID and OccNo represent the same occurrence, but only one is
necessary for classification
marsis_processed = marsis_processed_unique.drop(columns = 'OccNo')
```

```
#OccClassID and OccClassDisplayEng are identical, but only OccClassID
is integer encoded
marsis_processed = marsis_processed.drop(columns =
'OccClassDisplayEng')
```

```
#the same duplicate attribute (integer encoding vs non-encoded)
scenario applies to all attributes ending in '[...]DisplayEng'
marsis_processed.drop(marsis_processed.filter(regex='DisplayEng').colu
mns, axis=1, inplace=True)
```

3.2.4 - Removing Attributes Unfit for Encoding

Within the processed data frame, the reduced dimensions also include several attributes that cannot be encoded efficiently due to the contents of their records. Some variables that are unfit for encoding are the textual summaries of the occurrences, the reporting officers' names filing the occurrence, the textual descriptions of the nearest identifiable location to the occurrence location, and more. These attributes are subsequently dropped from the `marsis_processed` data frame.

```
#the dataset contains a set of attributes too varied or complex in
their contents to be encoded
#remove attributes unfit for encoding
unfit_for_encoding = ['IICName', 'Summary',
'NearestLocationDescription', 'OccDate', 'OccTime', 'OccDateTime',
'WindDirection']
```

```
marsis_processed = marsis_processed.drop(columns = unfit_for_encoding)
```

3.2.5 - Removing Attributes with Low Variance

In order to distill the dataset to its most relevant attributes, a low variance filter should also be applied to the `marsis_processed` data frame. Any attributes with variance beyond a certain threshold, or approximating zero variance, should be dropped. In the case of the study at hand, the numeric attributes' variance is calculated, stored in a list if they are approaching zero variance, and are then removed. Before the attributes are added to the removal list, they are manually contextualized with respect to the hypothesis to ensure no critical data is lost.

```
#determine variance of records per column in dataset
attribute_variance = marsis_processed.var(numeric_only =
True).to_frame()
```

```
#remove low variance attributes from dataset
low_variance_attributes = ['IncludedInDailyEnum',
'MajorChangesIncludedInDaily', 'LatEnum', 'LongEnum']
```

```
marsis_processed = marsis_processed.drop(columns =
low_variance_attributes)
```

3.2.6 - Removing Attributes Irrelevant to Hypothesis

The final step of this project's data cleansing is removing any attributes which may be irrelevant to the hypothesis under study. These are primarily attributes which serve an administrative purpose on when records were filed, closed, released, and published by the Transportation Safety Board. They are attributes which only exist for internal reporting or record identification purposes, and are to be dropped from the processed dataset.

```
#remove administrative attributes
administrative_attributes = ['ReleasedDate', 'OccClosedDate',
'EntryDate', 'ReportedDate', 'ReportedByID', 'OccID', 'TimeZoneID']
```

```
marsis_processed = marsis_processed.drop(columns =
administrative_attributes)
```

Section 4: Target Attribute Creation, and Building Train, Validation, Testing Sets

4.1 - Target Variable Creation

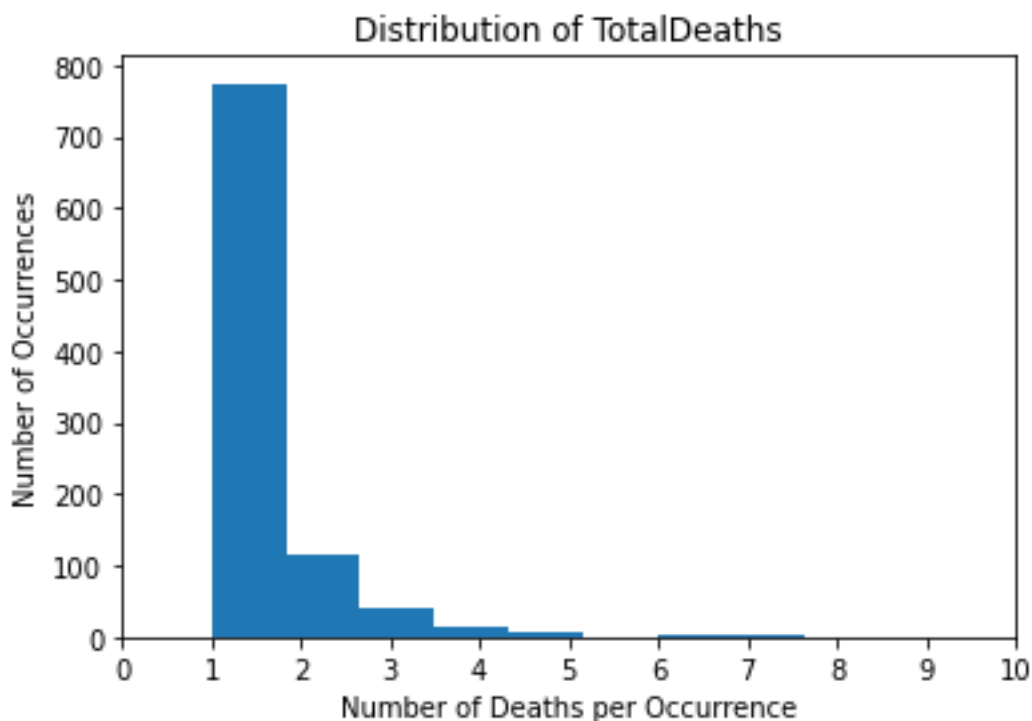
From the final `marsis_processed` dataset, one can infer the target variable to be `TotalDeaths`, but this assumption is not optimal for the hypothesis under study. The hypothesis aims to identify the deterministic factors in occurrences resulting in a marine fatality, and not to quantify the amount of deaths caused by marine occurrence attributes. This frames the study as one which identifies the classification of a marine fatality occurrence. Firstly, calculating the number of class levels required for fatality classification is required.

```

#store distribution of TotalDeaths
marsis_deadly = marsis_processed[marsis_processed['TotalDeaths'] > 0]

#the range of deaths is between 0, and 84
#most deaths are distributed between 0 and 5
plt.hist(marsis_deadly['TotalDeaths'], bins = 100)
plt.xticks(np.arange(0, max(marsis_deadly['TotalDeaths']), 1))
plt.title('Distribution of TotalDeaths')
plt.xlabel('Number of Deaths per Occurrence')
plt.ylabel('Number of Occurrences')
plt.xlim(0, 10)
plt.show()

```



```

#the average death count for fatal occurrences is 1.45 deaths
import statistics
statistics.mean(marsis_deadly['TotalDeaths'])

```

```

#since the number of average deaths is less than 2, the target
variable death classifier will be a binary Yes or No
#add the target variable OccDeathClassID with the classification 1 =
yes, 0 = no
marsis_processed['OccDeathClassID'] =
np.where(marsis_processed['TotalDeaths']!= 0, 1, 0)

```

4.2 - Data Splitting

With the target variable `OccDeathClassID` appended as a binary classification attribute to the `marsis_processed` data frame, the next step is to create input and output arrays for further splitting into training and test sets.

The input array will be a data frame composed of all the attributes under study, excluding the target variable and any of its direct attribute dependents. These attribute dependents are attributes which simulate or have an equivalent data value that strongly correlates to the target variable's classification.

```
#create input array (excluding target variable and its direct  
attribute dependents)  
x = marsis_processed.loc[:,  
~marsis_processed.columns.isin(['OccDeathClassID', 'TotalDeaths',  
'OccClassID', 'ImoClassLevelID'])]
```

The corresponding output array is a 1 dimensional data frame consisting only of the target variable, `'OccDeathClassID'`.

```
#create output array (including target variable)  
y = marsis_processed['OccDeathClassID'].to_frame()
```

With input and output arrays separating the target variables from the input variables, one can use sklearn's `model_selection` module to invoke a `train_test_split()` function to create training and testing data sets. Training sets are defined as `x_train` (with the input array attributes), and `y_train` (with the output array attribute). Similarly, the test sets are defined as `x_test` (with the input array attributes), and `y_test` (with the output array attribute). The split between training and testing sets is set at an 70/30 proportionality, with a stratified sampling technique whereby an approximately equal quantity of target attribute records are distributed amongst both training and testing sets.


```
import numpy as np
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    x, y,
    train_size = 0.7,
    test_size = 0.3,
    # random_state = 4,
    stratify = y)
```

One can verify with a subsequent command what the stratified sampling proportion approximates to. In this case, `y_train` has approximately 2.136% observations with a positive `OccDeathClassID`, and `y_test` has approximately 2.141% observations with a positive `OccDeathClassID`.

```
#verify proportionality of stratification in output array
y_train.dtypes
stratified_y_train = len(y_train[y_train['OccDeathClassID'] == 1]) /
len(y_train)
print(stratified_y_train)
```

```
Out[]: 0.021361000915471468
```

```
stratified_y_test = len(y_test[y_test['OccDeathClassID'] == 1]) /
len(y_test)
print(stratified_y_test)
```

```
Out[]: 0.021415889924545052
```

Section 5: Formal Dimensionality Reduction

Begin by loading Section 4 dependencies into the IDE:

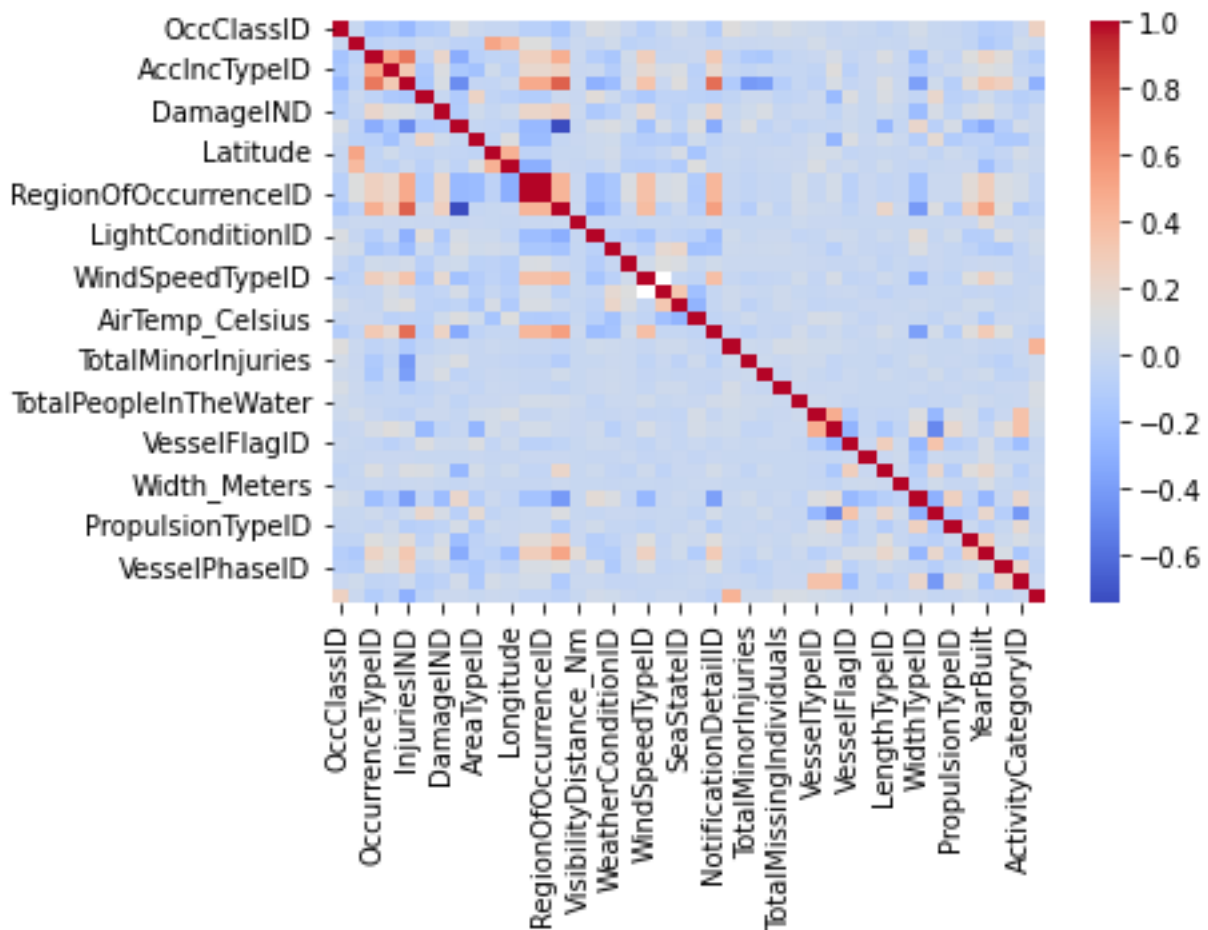
```
import xgboost
import numpy as np
import seaborn as sns
import sklearn.model_selection
from matplotlib import pyplot
from math import sqrt
```

5.1 - Removing Attributes with High Correlations

One of the selected methods of formal dimensionality reduction for this study is the identification and removal of highly correlated attributes in the already reduced feature set.

From the remaining attributes in the `marsis_processed` data frame, one can store the results of a correlation matrix to identify any directional relationships between the attributes. These directionalities are afterwards visualized as to easily identify them using a heatmap.

```
#verify if there are attributes that are too closely correlated and
are dependents of the target variable
attribute_correlation = marsis_processed.corr()
sns.heatmap(marsis_processed.corr(), fmt='.2g', cmap= 'coolwarm')
```



The strongest relationships identified include 'OccClassID' and 'TotalDeaths', which have already been removed from the training and testing split data. No further dimensionality reduction using this technique is appropriate given the results.

5.2 - Dimensionality Reduction by Random Forests Ensemble

In the processed dataset, many records contain NAs, and for some attributes' records, imputing them with a value may not make the most sense. To enable any further dimensionality reduction, the simplest way to identify the most important features is to use a modelling algorithm that supports missing values. Given that `scikitlearn` models are not capable of dealing with missing values, one can opt for XGBoost models to accomplish this task with potentially better computational efficiency.

Given the previously calculated proportionality of stratified sampling with respecting to `y_train` and `y_test` class levels, it is evident that the MARSIS dataset is very imbalanced. Out of all the records from 1975 to 2022, only 2% on average have an incident attached to them resulting in a maritime fatality. To compensate for this imbalance, the `XGBRFClassifier` will use the hyperparameter `scale_pos_weight` to give greater weight to the target variable's positive class. Ideally, the value of `scale_pos_weight` should correspond to the same proportionality of total negative samples to total positive samples. With this technique, one can mimic oversampling the data wherein the positive class is more prevalent. The hyperparameter `scale_pos_weight` will also be tweaked manually through the use of a floating point factor stored in a variable named `tweaking_param`.

```
#with XGBRFClassifier(), pass scale_pos_weight = x, where x is the  
total_negative_examples / total_positive_examples
```

```
#use tweaking param to adjust scale_pos_weight impact on False  
Positive and False Negative results  
tweaking_param = 0.45  
estimate = ((len(y_train[y_train['OccDeathClassID'] == 0])) /  
(len(y_train[y_train['OccDeathClassID'] == 1]))) * tweaking_param
```

An additional consideration that must be taken with the `XGBRFClassifier` is that it cannot support bootstrapping at each individual tree level in the random forest. To simulate bootstrapping, the subsample parameter in the model definition will serve to sample 80% of

the data specified - a sufficient amount to train the model on, while also introducing some variability into the ensemble.

Lastly, a consideration needs to be made for how the model will designate node split points within the random forest. The optimal node splitting heuristic parameter can be calculated by the following formula:

$$h = \frac{\sqrt{|features|}}{|features|}$$

```
#define the random forest ensembles model
heuristic_parameter = sqrt(len(marsis_processed.columns)) /
len(marsis_processed.columns)
model = xgboost.XGBRFClassifier(n_estimators = 100, subsample = 0.8,
colsample_bynode = heuristic_parameter, scale_pos_weight = estimate)
```

The performance of the model on its validation data will be performed via repeated k-fold cross validation, with three repeats across 10 folds.

```
#evaluate the model using repeated k-fold cross validation, with three
repeats and 10 folds
cv = sklearn.model_selection.RepeatedStratifiedKFold(n_splits=10,
n_repeats=3, random_state=1)
```

```
#evaluate the model and collect the scores
n_scores = sklearn.model_selection.cross_val_score(model, x_train,
y_train, scoring='accuracy', cv=cv, n_jobs=-1)
```

```
#evaluate model accuracy by taking the mean of the cross validation
scores, recording its standard deviation
print('Mean Accuracy: %.2f (%.2f)' % (np.mean(n_scores),
np.std(n_scores)))
```

```
Out[:]: 0.94 (0.00)
```

Prior to applying the `scale_pos_weight` hyperparameter, the model's accuracy was closer to 99%, indicative of overfitting. After applying an appropriate value for `scale_pos_weight`, the

model's accuracy is at 94% on the validation set, but its accuracy score alone cannot determine its ability to predict skillfully.

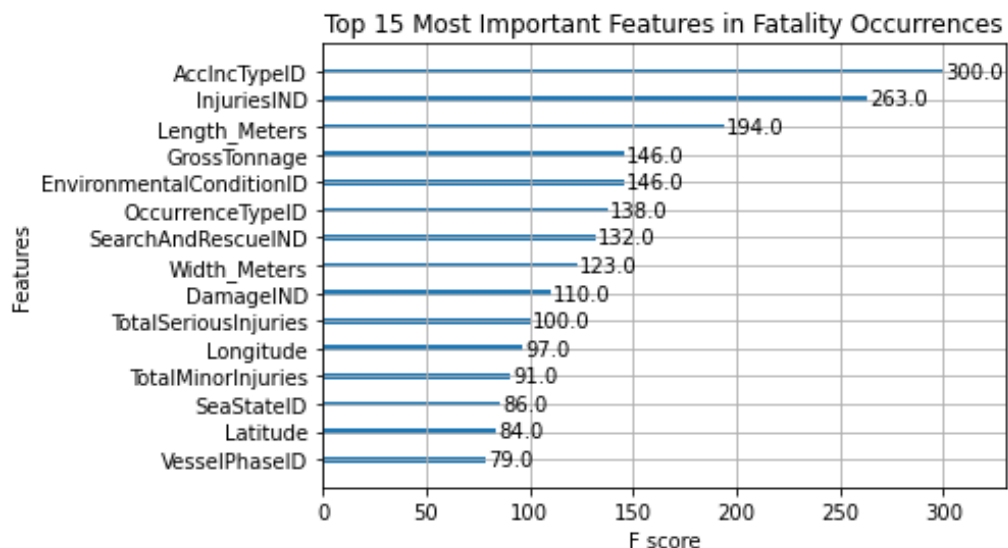
The model is ready to be fit to the training datasets using the following command:

```
#fit the random forests model to the training data
model.fit(x_train, y_train)
```

Having trained the model on the created training data sets, one is now able to extract the most important features contributing to an OccDeathClassID level. This is done by creating an importance matrix where the metric under consideration is the information gain of splits using each respective feature. This importance matrix is then visualized, showing the top 15 most importance features contributing to marine fatalities. The full importance matrix is also made available in a data frame named `feature_importance_df`.

```
#store feature importance
feature_important = 
model.get_booster().get_score(importance_type='gain')
keys = list(feature_important.keys())
values = list(feature_important.values())
feature_importance_df = pd.DataFrame(data=values, index=keys,
columns=["score"])
```

```
#visualize feature importance
xgboost.plot_importance(model, title = 'Top 15 Most Important Features
in Fatality Occurrences', max_num_features = 15)
pyplot.show()
```



Section 6: Model Building and Model Comparisons

6.1 - Random Forests Ensemble: Model Building and Evaluation

With the model built out and its most important features identified, one can test it on the previously created testing data sets. The next step is to pass the model the test data set and report on its accuracy.

```
#make predictions for test data and evaluate
from sklearn.metrics import accuracy_score

y_pred = model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Out[:]: Accuracy: 94.53%
```

For this run, the model maintains its same degree of accuracy as per the training data set. In addition to measuring accuracy, one can also create a classification report of the model's performance using `sklearn.metrics`.

```
#test and pred results
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
results_comparison = x_test.assign(target = y_test.values, prediction
= y_pred)
```

	precision	recall	f1-score	support
0	1.00	0.94	0.97	8819
1	0.28	0.97	0.43	193
accuracy			0.95	9012
macro avg	0.64	0.96	0.70	9012
weighted avg	0.98	0.95	0.96	9012

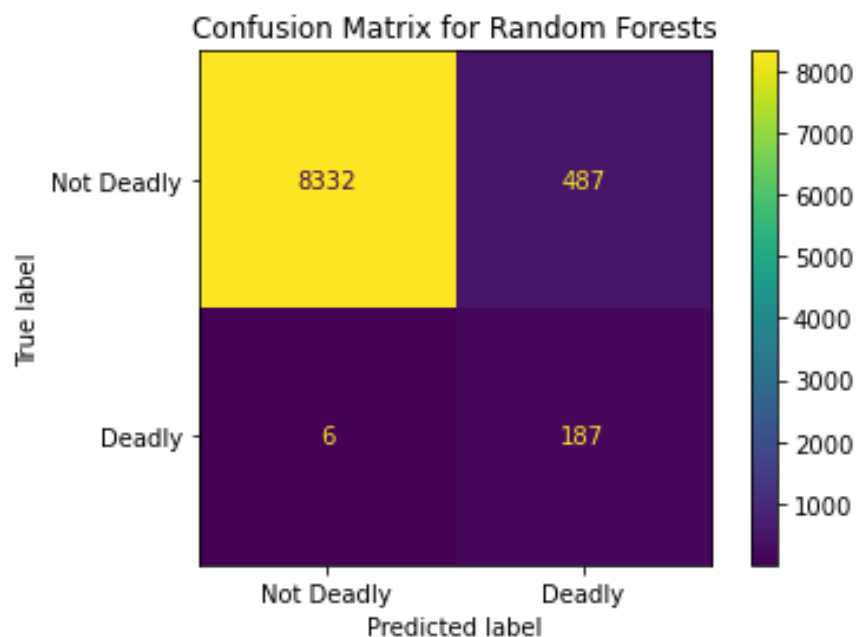
Classification report created for the manually weighted random forests ensemble.

From the manually weighted random forests classification report, one evaluates the model using the following criterion:

- The overall accuracy is strong, with 95% of instances predicted correctly by the model. However, given that the overall data set is very imbalanced, this is not the correct metric to be observing as a primary measure of success. Precision of positive classes is more important in measuring the overall performance of the model.
- The model's precision score is maximized at 100% for negative classes, but its precision struggles for positive classes at 28%. Looking at the visualized confusion matrix below, one can infer that the model is returning many results, but with a great volume of incorrectly identified false-positives.
- The model has a high recall percentage of 97% on the positive class, whereby a large majority of the real positive cases are predicted as positives.

One can visualize the impact of the precision and recall metrics in a confusion matrix using `pyplot`.

```
#visualize test and pred results
disp = sklearn.metrics.ConfusionMatrixDisplay.from_predictions(
    y_test,
    y_pred,
    display_labels= ['Not Deadly', 'Deadly'])
disp.ax_.set_title('Confusion Matrix for Random Forests')
plt.show()
```

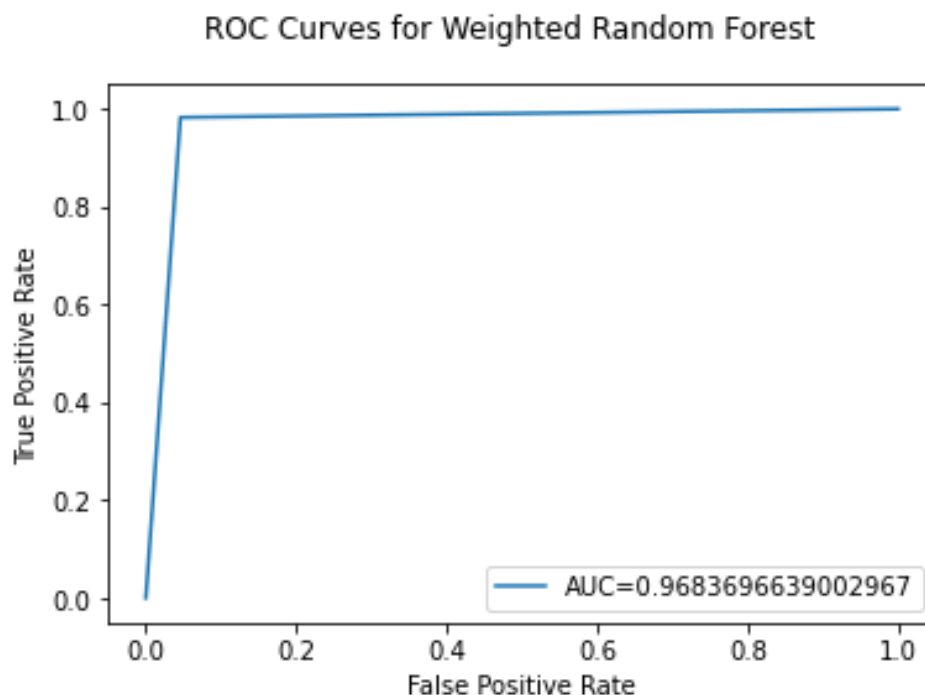


In the case of predicting marine occurrences that are likely to occur in deaths, having false positives may be indicative of a more overzealous model - something not to be seen as a negative, as it may prevent fatal marine accidents from occurring. Given the circumstances under study, false negatives are much more costly, potentially costing lives if disproportionate to true positives. Overall, the model's performance is satisfactory, and is able to identify deadly and not deadly scenarios effectively, while identifying false positives 5.4% of the time.

Lastly, the model's receiver operating characteristics (ROC) curve is plotted, with calculated area under curve (AUC) values appended to the visualized plot.

```
#visualize ROC AUC
fpr, tpr, _ = sklearn.metrics.roc_curve(y_test, y_pred)
auc = sklearn.metrics.roc_auc_score(y_test, y_pred)

plt.plot(fpr, tpr, label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.suptitle('ROC Curves for Weighted Random Forest')
plt.show()
```



The ROC curve of the manually weighted random forests ensemble model.

From the generated ROC curve, one can deduce that the model has a good measure of separability, while introducing type 1 and type 2 errors about 4% of the time. This is a good measure of the overall model's performance, distinguishing correctly between positive and negative `OccDeathClassID` instances approximately 96% of the time.

6.1.1 - Redefining the Random Forests' Optimal Weighting using Grid Search

In section 6.1, the model was tested upon a configuration where the hyperparameter `scale_pos_weight` was set to a value based on manual experimentation in an attempt to correctly weight the balance of positive class incidents. However, this manually selected value may not be the optimal value for the model's performance, and it needs to be checked against an algorithmically selected weighting value.

While the initial configuration leaned on the formula of `scale_pos_weight = total_negative_examples / total_positive_examples`, another approach to determine weighting values involves the use of a grid search. With this configuration, the model adopts weightings based on a grid of specified parameter values.

```
# 5.1.1 - Redefine the Model's Optimal Weighting using Grid Search
from sklearn.model_selection import GridSearchCV
```

```
#define grid weights and parameters
weights = [0.1, 0.25, 0.5, 0.75, 1, 10, 25, 50, 75, 100]
param_grid = dict(scale_pos_weight=weights)
```

The model is then evaluated using the same k-fold cross validation parameters as seen in section 5.2.

```
#evaluate the model using repeated k-fold cross validation, with three
repeats and 10 folds
cv = sklearn.model_selection.RepeatedStratifiedKFold(n_splits=10,
n_repeats=3, random_state=1)
```

The model is then iteratively evaluated using the defined grid search parameters, this time scoring itself on the area under its receiver operating characteristic curve.

```
#define grid search execution
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1,
cv=cv, scoring='roc_auc')
grid_result = grid.fit(x_train, y_train)
```

The optimal configuration is printed to the console, displaying the best AUC score, along with the grid parameter values that allowed the model to reach that best AUC score. Iteratively, the rest of the parameter values are evaluated in the model, printing out their respective results to the console.

```
#report the best configuration
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
```

```
# report all configurations
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.993950 using {'scale_pos_weight': 10}
0.993501 (0.001737) with: {'scale_pos_weight': 0.1}
0.993186 (0.001682) with: {'scale_pos_weight': 0.25}
0.993143 (0.001689) with: {'scale_pos_weight': 0.5}
0.993123 (0.001708) with: {'scale_pos_weight': 0.75}
0.993190 (0.001628) with: {'scale_pos_weight': 1}
0.993950 (0.001792) with: {'scale_pos_weight': 10}
0.993884 (0.001904) with: {'scale_pos_weight': 25}
0.993271 (0.002131) with: {'scale_pos_weight': 50}
0.992474 (0.002424) with: {'scale_pos_weight': 75}
0.991679 (0.002638) with: {'scale_pos_weight': 100}
Accuracy: 98.74%
```

The results of each iterative model evaluation given the specified grid search parameters for the scale_pos_weight hyperparameter.

In this run, it appears that the best scale_pos_weight determined was a value of 10 from the pre-specified grid parameter values. The model is now redefined based off the weightings to account for this optimal hyperparameter value:

```
#redefine the model given best defined weighting
model = xgboost.XGBRFClassifier(n_estimators = 100, subsample = 0.8,
scale_pos_weight = (list(grid_result.best_params_.values())[0]))
```

The model is then subsequently fit on the training data, and evaluated on the test sets created earlier, with the exact same procedure displayed in section 6.1.

```
#fit the random forests model to the training data
model.fit(x_train, y_train)
```

```
#make predictions for test data and evaluate
from sklearn.metrics import accuracy_score
y_pred = model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
#test and pred results
#accuracy score
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
#classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
results_comparison = x_test.assign(target = y_test.values, prediction
= y_pred)
```

```
Accuracy: 98.97%
0.9897174138186122
      precision    recall  f1-score   support

     0       1.00      0.99      0.99     13229
     1       0.69      0.94      0.80        289

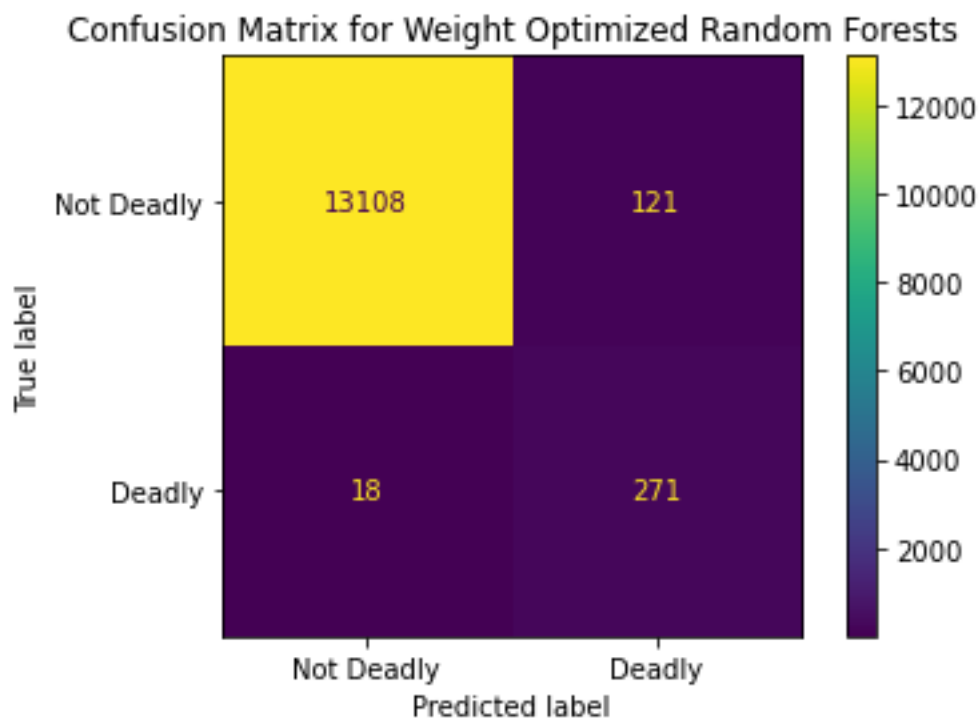
 accuracy          0.99          0.99          0.99     13518
 macro avg          0.84          0.96          0.90     13518
 weighted avg          0.99          0.99          0.99     13518
```

The generated classification report for the optimally-weighted random forests ensemble model.

The generated classification report displays a 98.97% accuracy based on the test data sets created. This is an improvement from the 94.53% reported in the manually-weighted variant of the model. More importantly, the recall metric for positive classes has increased substantially,

from 0.28 in the manual model, to 0.69 in the optimally-weighted model. This is a large increase in the amount of predictions of positive classes correctly identified. However, they do not come without a cost, and the following visualizations show why the type 1 and type 2 errors are more concerning in this optimally-weighted model.

```
#visualize test and pred results
disp = sklearn.metrics.ConfusionMatrixDisplay.from_predictions(
    y_test,
    y_pred,
    display_labels= ['Not Deadly', 'Deadly'])
disp.ax_.set_title('Confusion Matrix for Weight Optimized Random
Forests')
plt.show()
```



The generated confusion matrix for the optimally-weighted random forests model.

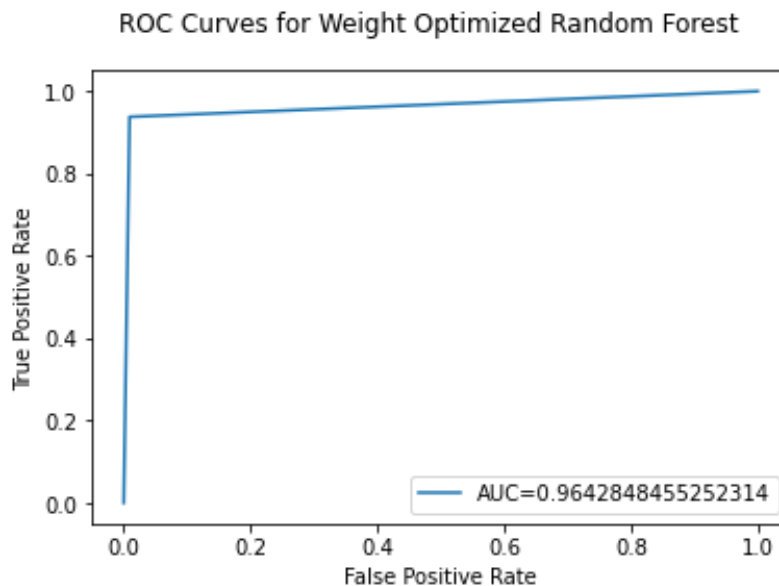
As per the generated classification report, one can see that the overall volume of true positive predictions made by the optimally-weighted model is much higher. In addition, there is a substantial decrease in the amount of false positives reported - a decrease of 75% in false

positives. This improvement is accompanied by a 200% increase in the amount of false negatives reported, making the model much more dangerous if it were pushed into production.

Having a false positive is acceptable in the model because if an occurrence is predicted as deadly, but in fact is not, there is no loss of life associated with it. There are other considerations with a higher false positive prediction rate, such as the time and monetary cost in sending out coast guard personnel to verify the incident and make sure no one is under deadly peril. In this case, the primary ethical consideration ought to heavily penalize false negatives, by which no alarm is sounded for a deadly incident, and lives are put at risk for the sake of saving time, money, or other resource.

```
#visualize ROC AUC
fpr, tpr, _ = sklearn.metrics.roc_curve(y_test, y_pred)
auc = sklearn.metrics.roc_auc_score(y_test, y_pred)

plt.plot(fpr, tpr, label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.suptitle('ROC Curves for Weight Optimized Random Forest')
plt.show()
```



The generated ROC AUC plot for the optimally-weighted random forests model.

6.1.2 - Creating a Balanced Random Forests Algorithm

While the previously created random forests model attempted to account for the data imbalance using the `scale_pos_weight` hyperparameter, one more approach that may yield viable results is adjusting the algorithm that accounts for the data imbalance out of the box. This adjustment comes in the form of the `BalancedRandomForestClassifier` class originating from the `imbalanced-learn` library package available via PyPi¹². The classifier model in this case resamples the data under study in order to directly reflect the class distribution for `OccDeathClassID`.

Notably, this model's algorithm is incapable of dealing with NA values unlike XGBoost. Given this distinction, the remaining NA values in the initial training and testing data sets are saved to variant variables identifying that they are imputed value variables. Both test and training NAs are imputed with the mean of the values per attribute.

```
##since imbalanced-learn models cannot handle NANs like XGB00ST, use
an imputation method for NANs
learning_sets = [x_train, y_train]
for set in learning_sets:
    set.apply(pd.to_numeric, errors = 'coerce')
test_sets = [x_test, y_test]
for set in test_sets:
    set.apply(pd.to_numeric, errors = 'coerce')

imp_x_train = x_train.copy(deep=True)
imp_x_train.fillna(imp_x_train.mean(), inplace = True)

imp_x_test = x_test.copy(deep=True)
imp_x_test.fillna(imp_x_test.mean(), inplace = True)

imp_y_train = y_train.copy(deep=True)
imp_y_train.fillna(imp_y_train.mean(), inplace = True)
imp_y_train = np.ravel(imp_y_train)

imp_y_test = y_test.copy(deep=True)
imp_y_test.fillna(imp_y_test.mean(), inplace = True)
imp_y_test = np.ravel(imp_y_test)
```

¹² Lemaitre, G., & Aridas, C. (2022, May 22). *imbalanced-learn 0.9.1*. PyPi. Retrieved from <https://pypi.org/project/imbalanced-learn/>

Once imputation of mean values is complete, the package is imported with the balanced classifier model. The imputed training and testing data sets are used for this implementation, and the results are visualized with the same procedures shown in section 6.1 and 6.1.1.

```
#evaluate the model using repeated k-fold cross validation, with three
repeats and 10 folds
cv = sklearn.model_selection.RepeatedStratifiedKFold(n_splits=10,
n_repeats=3, random_state=1)

#evaluate the model and collect the scores
scores = sklearn.model_selection.cross_val_score(model, imp_x_train,
imp_y_train, scoring='roc_auc', cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % np.mean(scores))

#fit the random forests model to the training data
balanced_model.fit(imp_x_train, imp_y_train)

#make predictions for test data and evaluate
y_balanced_pred = balanced_model.predict(imp_x_test)
balanced_accuracy = accuracy_score(imp_y_test, y_balanced_pred)
print("Accuracy: %.2f%%" % (balanced_accuracy * 100.0))

#test and pred results
#accuracy metrics
from sklearn.metrics import accuracy_score
print(accuracy_score(imp_y_test, y_balanced_pred))

#classification report
from sklearn.metrics import classification_report
print(classification_report(imp_y_test, y_balanced_pred))
```

```
Accuracy: 93.99%
0.9398579671549046
      precision    recall  f1-score   support

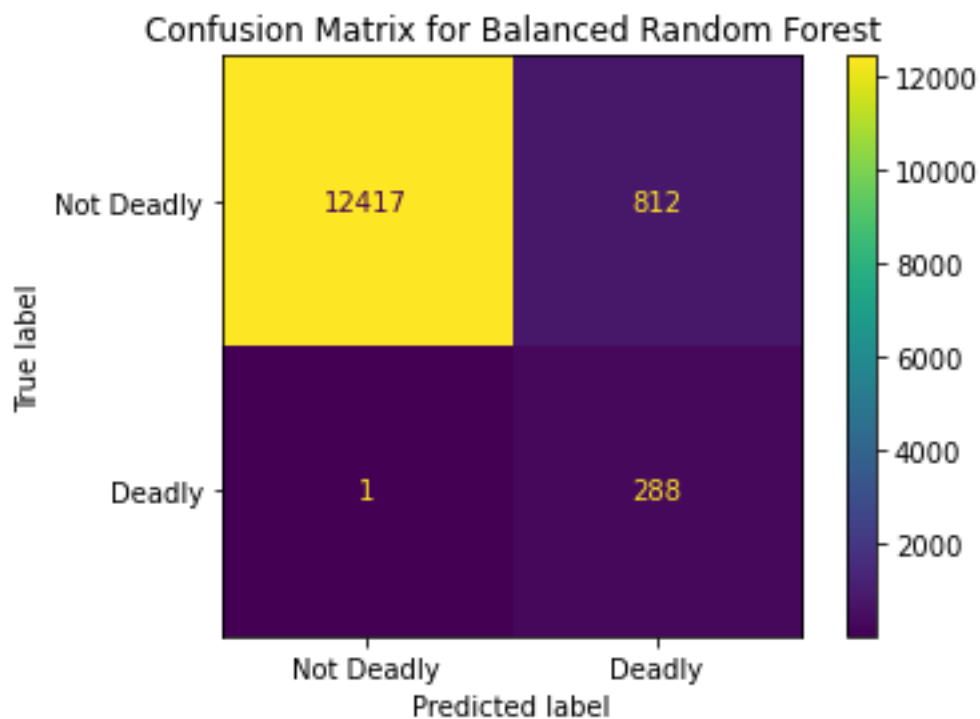
     0       1.00      0.94      0.97     13229
     1       0.26      1.00      0.41        289

 accuracy          0.94     13518
 macro avg         0.63      0.97      0.69     13518
 weighted avg         0.98      0.94      0.96     13518
```

The generated classification report for the balanced random forests model.

This model is very similar in its performance to the manually selected weighting variant created in section 6.1. Accuracy, recall, and precision metrics are on-par with the manual model, with potentially similar weighting values being assigned to both balanced and manually-weighted models. The model's performance is subsequently visualized using the same procedures in 6.1 and 6.1.1.

```
#visualize test and pred results
disp = sklearn.metrics.ConfusionMatrixDisplay.from_predictions(
    imp_y_test,
    y_balanced_pred,
    display_labels= ['Not Deadly', 'Deadly'])
disp.ax_.set_title('Confusion Matrix for Balanced Random Forest')
plt.show()
```

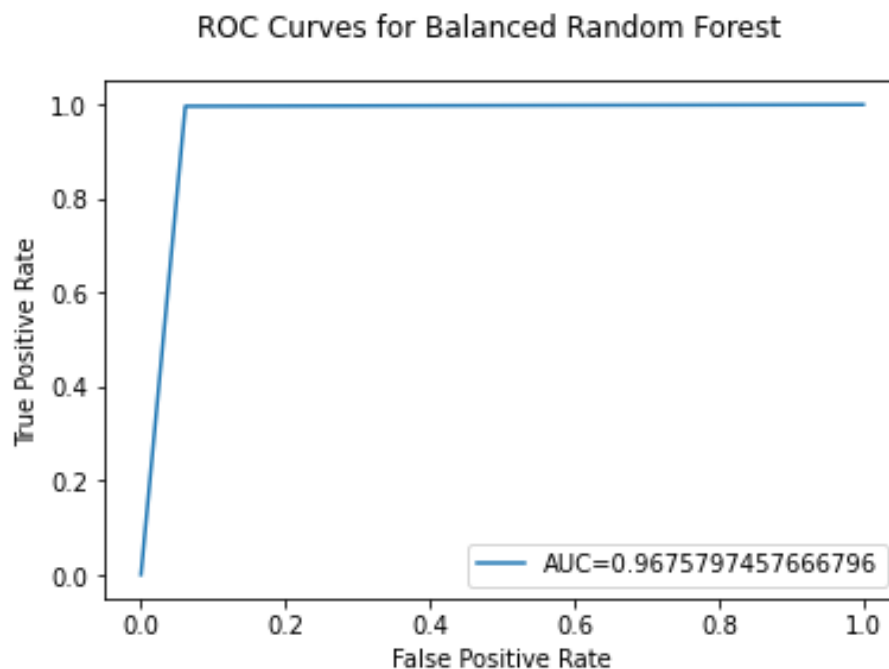


The generated confusion matrix for the balanced random forests model.

The balanced model is much more overzealous in its classification of false negatives than the manually weighted model variant. This comes at the expense of a large increase of 66.7% in false positive predictions, which in the case of marine safety, may be difficult to justify the costs associated with the model's use in production. Out of the variant models created thus far, this variant is the one that is the most safe, and minimizes the amount of deadly incidents which could be ignored due to classification errors.

```
#visualize ROC AUC
fpr, tpr, _ = sklearn.metrics.roc_curve(imp_y_test, y_balanced_pred)
auc = sklearn.metrics.roc_auc_score(imp_y_test, y_balanced_pred)

plt.plot(fpr, tpr, label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.suptitle('ROC Curves for Balanced Random Forest')
plt.show()
```



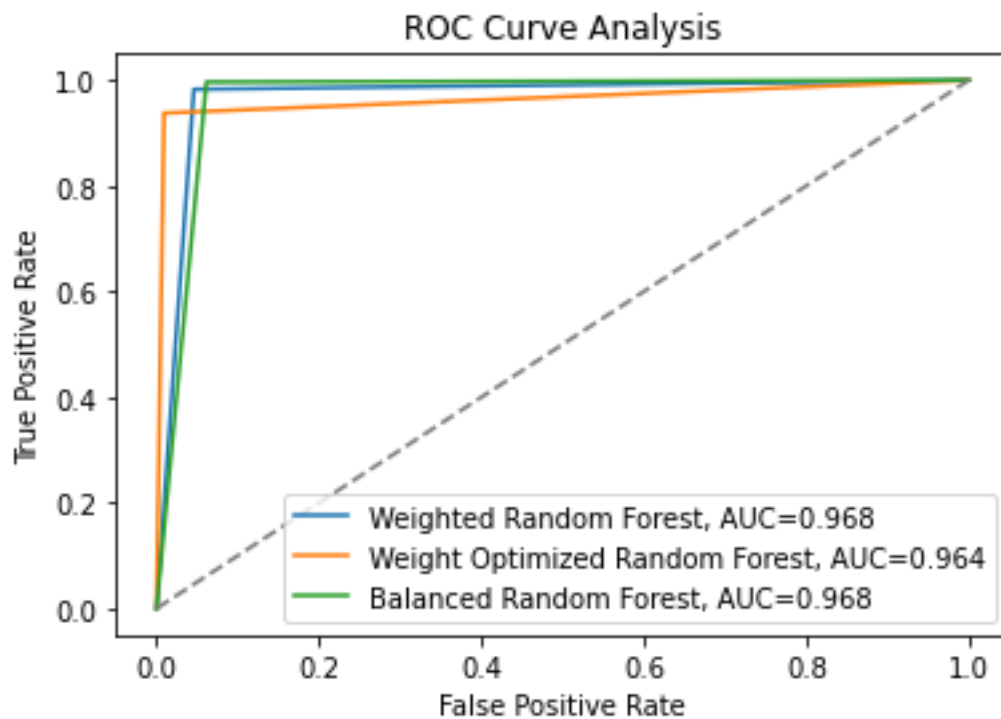
The generated ROC AUC plot for the balanced random forests model.

6.1.3 - Comparison of Random Forest Model Variants

To conclude the analysis of the random forest model variants, one can plot the compiled ROC AUC curves for each respective model and compare their performance given the context of the study.

```
#plot comparison figure
for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
             label="{}, AUC={:.3f}".format(result_table.loc[i]
['classifiers'], result_table.loc[i]['auc']))

plt.plot([0,1], [0,1], color='grey', linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title('ROC Curve Analysis')
plt.legend(loc=4)
plt.show()
```



The combined ROC AUC plots for the random forest models under study.

From the results displayed in the ROC Curve Analysis, all three models are performant with respect to overall accuracy and ability to discern positive classifications from negative classifications in predictions effectively. The weight-optimized random forests model is the most dangerous of the group to implement in actuality due to its propensity to not penalize false negatives as strictly; potentially resulting in fatal marine occurrences not being identified as being deadly.

Conversely, both the manually weighted random forest and the balanced random forest models are very similar, and even share the same AUC score values. Despite their similarities, the balanced random forest algorithm tends to be overzealous in its penalization of false negatives, skewing the overall predictions heavily towards a greater volume of false positives. While this affects the recall of the model in a negative way - and is likely best for minimizing as many potentially fatal occurrences from being overlooked - it is improbable that this model would be implemented by the Transportation Safety Board of Canada. With the amount of false positives returned by the balanced random forest model, type-1 errors would cost the TSB in the long term. Too many instances of false positives would require marine staff to investigate many occurrences without in a fatality, and by continuously making an error, it is likely that the model's trust would be eroded, despite saving lives through its application. As such, it appears the best random forest model available under study is the manually weighted one.

6.2 - Weighted Logistic Regression: Model Building and Evaluation

In order to contrast against the results coming from a more complicated model structure such as random forests, a logistic regression model is also experimented with in order to derive an answer to the hypothesis under study. The logistic regression algorithm used as part of this experiment will be one originating from the `scikitlearn` package. Similarly to the previously enacted balanced random forests classifier, the logistic regression algorithm is incapable of dealing with NA values, and consequently will also use the imputed test and train variables previously declared in section 6.1.2.

The same issue of imbalanced data persists with the logistic regression model, whereby class weighting parameters must be assigned to account for the imbalance. The hyperparameter under study for the logistic regression model in `scikitlearn` is `class_weight`, where a dictionary value is passed specifying the class weightings when evaluating the model. The heuristic value applied to `class_weight` is the inverse of the class distribution of the data set at hand. In the case of the segmented training and test data, this involves a relative class weighting ratio of 2.1361 to the negative `OccDeathClassID` paired with a relative class weighting ratio of 97.8639 to the positive `OccDeathClassID`.

Notably, given the amount of variables as part of both the training and test data sets, evaluation of the model may fail to converge due to an increased volume of ill-fitting observations. As a mitigating parameter, the maximum number of iterations for the logistic regression before a maximum likelihood estimate exists is capped at 10,000 iterations, increasing computation time quite significantly.

```
#define the logistic regression model
#account for unbalanced dataset as in XGB00ST using class_weight
hyperparameter
w = {0:2.1361, 1:97.8639}
log_model = LogisticRegression(random_state=4, class_weight=w,
max_iter=10000)
```

The model may throw convergence warnings, which can be hidden from the console using the following command:

```
#optional: hide ConvergenceWarnings for logistic regression output
import warnings
from sklearn.exceptions import ConvergenceWarning

with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=ConvergenceWarning)
```

Afterwards, the same method of evaluation and visualization is applied to the weighted logistic regression model as seen in the random forests variants in sections 6.1, 6.1.1, and 6.1.2.

```

#evaluate the model using repeated k-fold cross validation, with three
repeats and 10 folds
cv = sklearn.model_selection.RepeatedStratifiedKFold(n_splits=10,
n_repeats=3, random_state=1)

#evaluate the model and collect the scores
n_log_scores = sklearn.model_selection.cross_val_score(log_model,
imp_x_train, imp_y_train, scoring='accuracy', cv=cv, n_jobs=-1)

# report performance
print('Mean Accuracy: %.2f (%.2f)' % (np.mean(n_log_scores),
np.std(n_log_scores)))

#fit the model to the training data
log_model.fit(imp_x_train, imp_y_train)

#make predictions for test data and evaluate
y_log_pred = log_model.predict(imp_x_test)
log_accuracy = accuracy_score(imp_y_test, y_log_pred)
print("Accuracy: %.2f%%" % (log_accuracy * 100.0))

#test and pred results
#accuracy metrics
from sklearn.metrics import accuracy_score
print(accuracy_score(imp_y_test, y_log_pred))

#classification report
from sklearn.metrics import classification_report
print(classification_report(imp_y_test, y_log_pred))

```

```

Accuracy: 86.15%
0.8615179760319573
      precision    recall  f1-score   support

     0       1.00      0.86      0.92     13229
     1       0.13      0.92      0.22        289

 accuracy          0.86     13518
 macro avg         0.56     13518
 weighted avg         0.98     13518

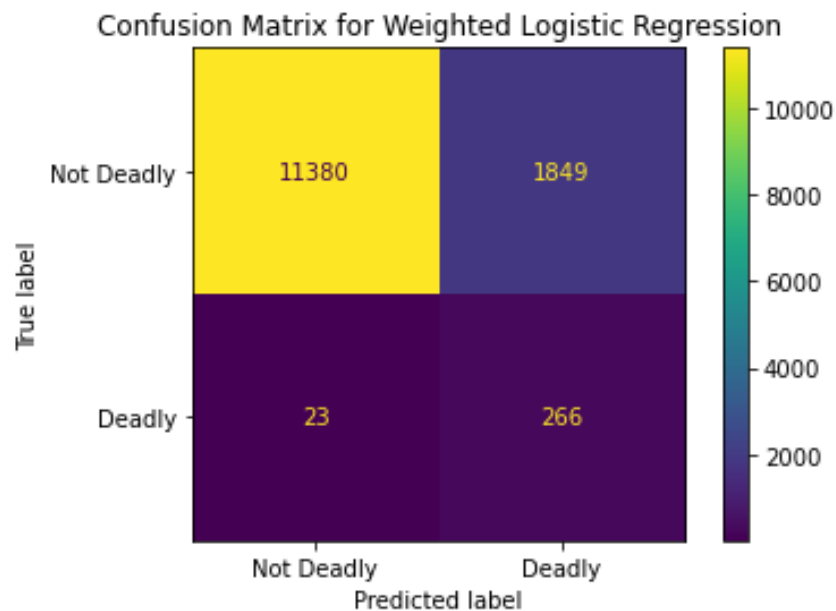
```

The generated classification report for the weighted logistic regression model.

The overall accuracy for the model concludes with a value of 86.15%, notably worse than any of the afore tested random forests algorithms. Precision for the positive class

struggles significantly, with an approximate 50% drop in precision score when compared to the random forests algorithms.

```
#visualize test and pred results
disp = sklearn.metrics.ConfusionMatrixDisplay.from_predictions(
    imp_y_test,
    y_log_pred,
    display_labels= ['Not Deadly', 'Deadly'])
disp.ax_.set_title('Confusion Matrix for Logistic Regression')
plt.show()
```



The generated confusion matrix for the weighted logistic regression model.

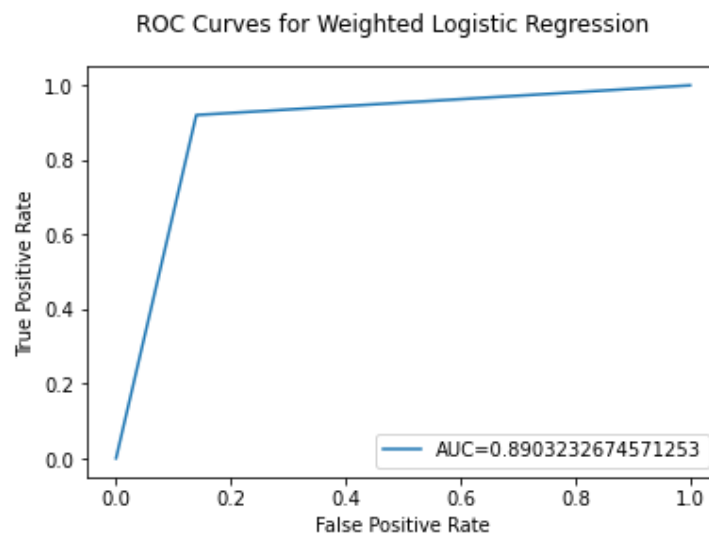
As expected, the logistic model's drop in recall also suggests that it makes more type 1 and type 2 errors. Fortunately, a lot of the errors made by the model are type 1 errors. Influenced heavily by the imbalanced data under consideration, the logistic regression skews towards the result most associated with the most common class in `OccDeathClassID`. To conclude, with many errors on both false positive and false negative ends, this model cannot compare in performance to the random forests created earlier, and manages a respectable AUC value of 0.89.

```

#visualize ROC AUC
fpr, tpr, _ = sklearn.metrics.roc_curve(imp_y_test, y_log_pred)
auc = sklearn.metrics.roc_auc_score(imp_y_test, y_log_pred)

plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.suptitle('ROC Curves for Weighted Logistic Regression')
plt.show()

```



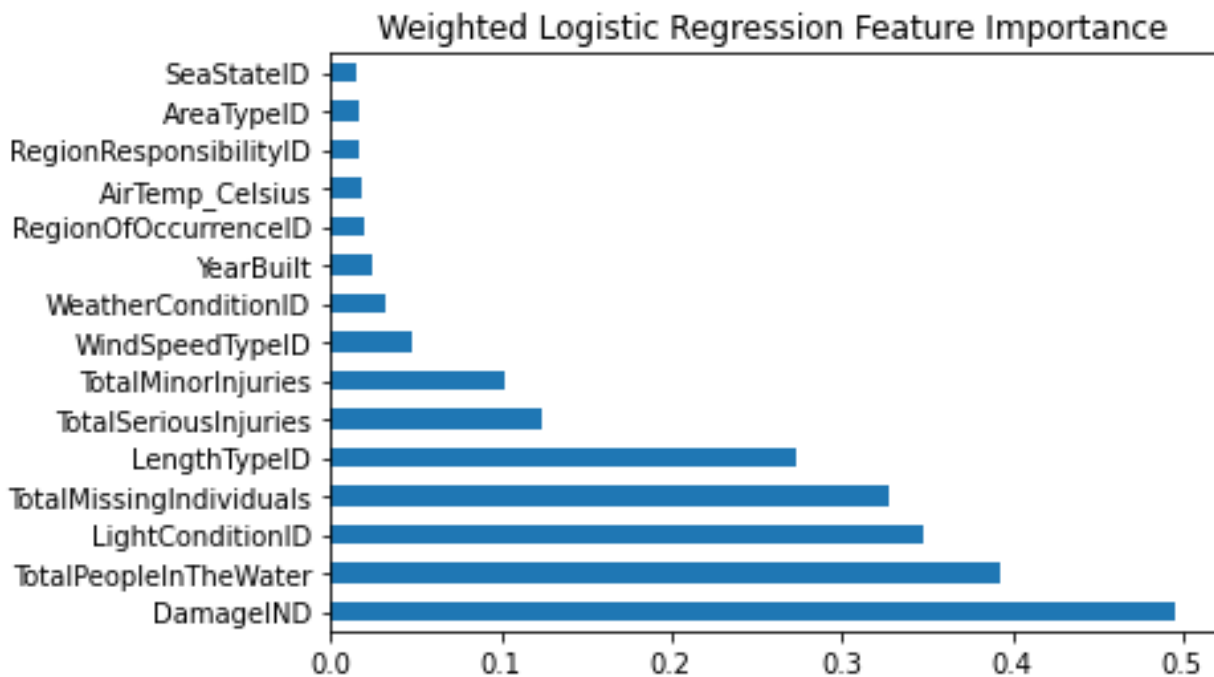
The generated ROC AUC plot for the weighted logistic regression model.

Lastly, the feature importance of the logistic regression classifier is calculated and visualized using the following command:

```

#logistic regression variable importance
log_importance = pd.Series(log_model.coef_[0], index =
imp_x_test.columns)
log_importance = log_importance.sort_values(ascending=False)
log_importance.nlargest(15).plot(kind = 'barh', title = 'Weighted
Logistic Regression Feature Importance')

```



Top 15 most important features by logistic regression coefficient.

6.3 - Naive Bayes Classifier: Model Building and Evaluation

Lastly, the final model under experimentation is a Naive Bayes classifier, also originating from the `scikitlearn` package. Given the assumption of Naive Bayes classifiers to assume input variables as being independent of other input variables, this model is expected to perform the poorest out of the selected models. Despite this, a predictive model of this type was taken into consideration to provide diversity in the techniques under study, and to subsequently analyze the efficacy of a conditional probability model given the original data set.

Similarly to the previously enacted weighted logistic regression model, the Naive Bayes algorithm is incapable of dealing with NAN values, and consequently will also use the imputed test and train variables previously declared in section 6.1.2.

```
from sklearn.naive_bayes import GaussianNB

#define the naive bayes model
nb_model = GaussianNB()
```


The same method of evaluation and visualization is applied to the Naive Bayes model as seen in sections 6.1, 6.1.1, 6.1.2, and 6.2.

```
#evaluate the model using repeated k-fold cross validation, with three
repeats and 10 folds
cv = sklearn.model_selection.RepeatedStratifiedKFold(n_splits=10,
n_repeats=3, random_state=1)

#evaluate the model and collect the scores
n_nb_scores = sklearn.model_selection.cross_val_score(nb_model,
imp_x_train, imp_y_train, scoring='accuracy', cv=cv, n_jobs=-1)

# report performance
print('Mean Accuracy: %.2f (%.2f)' % (np.mean(n_nb_scores),
np.std(n_nb_scores)))

#fit the model to the training data
nb_model.fit(imp_x_train, imp_y_train)

#make predictions for test data and evaluate
y_nb_pred = nb_model.predict(imp_x_test)
nb_accuracy = accuracy_score(imp_y_test, y_nb_pred)
print("Accuracy: %.2f%%" % (nb_accuracy * 100.0))

#test and pred results
from sklearn.metrics import classification_report
print(classification_report(imp_y_test, y_nb_pred))
```

```
Mean Accuracy: 0.47 (0.15)
Accuracy: 43.46%
          precision    recall  f1-score   support

     0       0.99      0.43      0.60     13229
     1       0.03      0.80      0.06        289

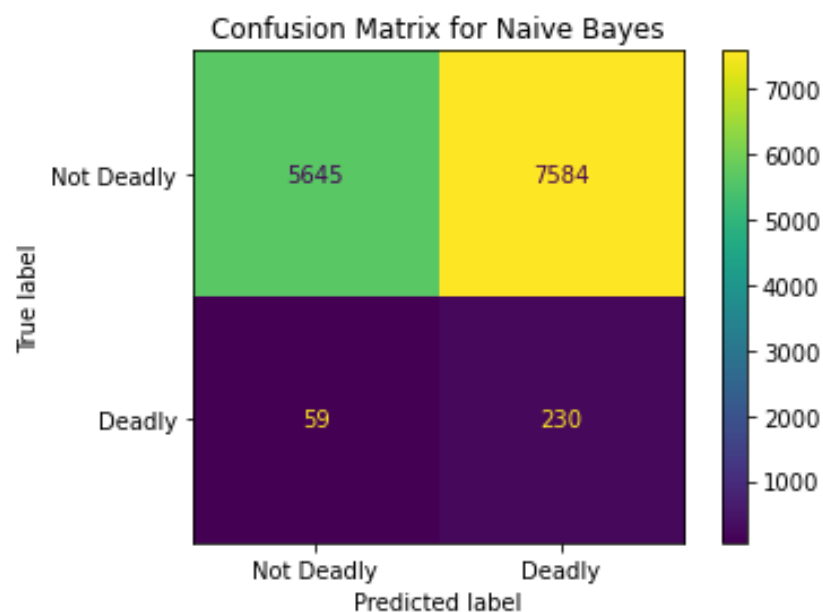
 accuracy          0.43     13518
 macro avg          0.51     13518
 weighted avg       0.97     13518
```

The generated classification report for the Naive Bayes model.

Given the complexity of the number of variables as part of the training and testing data sets, the Naive Bayes classifier was not expected to perform better than the afore tested models under study. There are likely interactions between the variables that are not identified in

the model, and the sheer volume of variables as part of the training and testing data sets makes this model struggle with recall, precision, and accuracy. With regards to precision for the positive class, it performs exceptionally poorly and one is better off guessing the results of the marine occurrence rather than relying on this model. This poor performance is supported by the ROC AUC score of 0.611, just marginally higher than the 0.5 threshold whereby the classifier cannot distinguish between positive and negative class associations.

```
#visualize test and pred results
disp = sklearn.metrics.ConfusionMatrixDisplay.from_predictions(
    imp_y_test,
    y_nb_pred,
    display_labels= ['Not Deadly', 'Deadly'])
disp.ax_.set_title('Confusion Matrix for Naive Bayes')
plt.show()
```



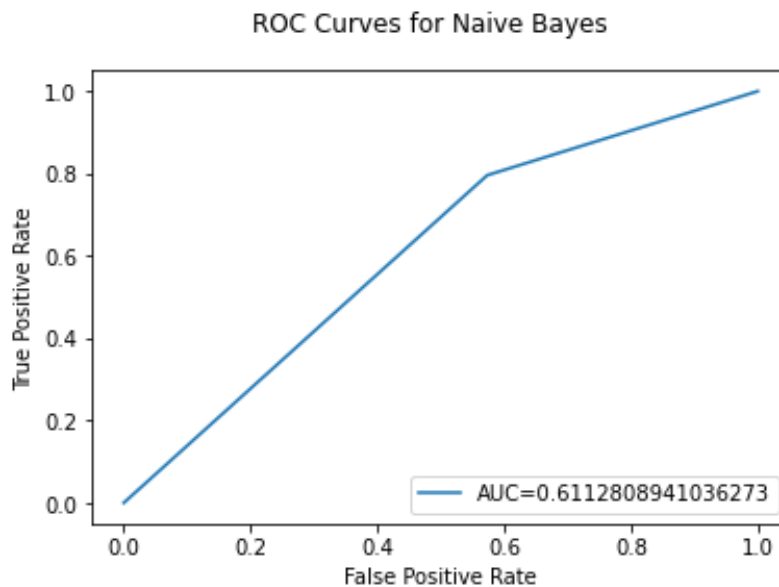
The generated confusion matrix for the Naive Bayes model.

```

#visualize ROC AUC
fpr, tpr, _ = sklearn.metrics.roc_curve(imp_y_test, y_nb_pred)
auc = sklearn.metrics.roc_auc_score(imp_y_test, y_nb_pred)

plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.suptitle('ROC Curves for Naive Bayes')
plt.show()

```



The generated ROC AUC plot for the Naive Bayes model.

Given the poor performance of the Naive Bayes model, its feature importance data will not be calculated and will not be taken into further consideration.

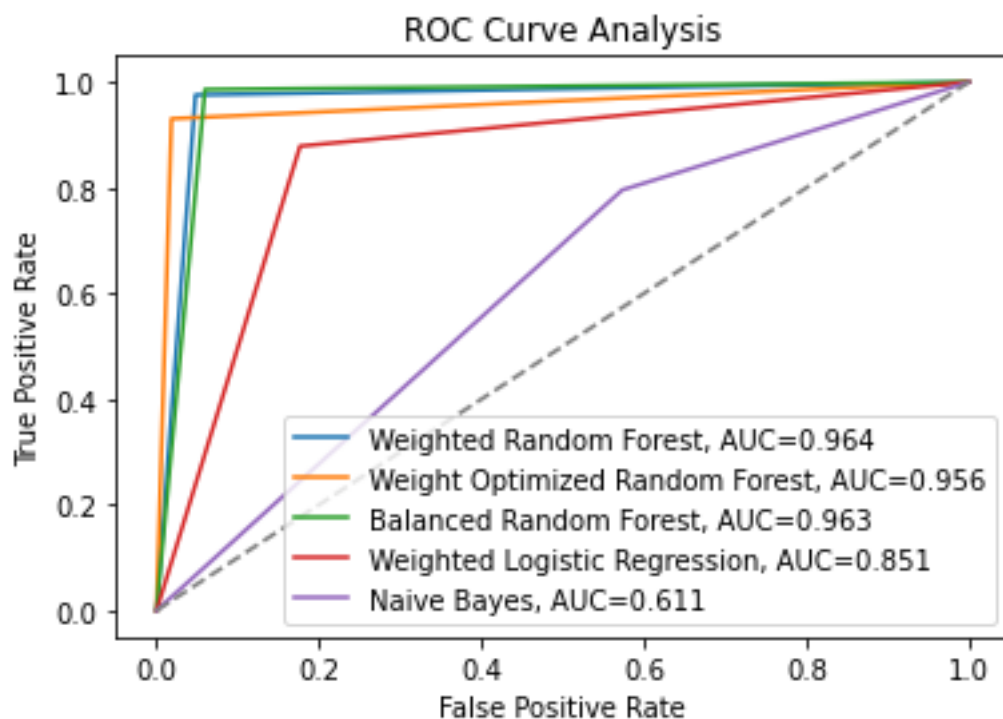
Section 7: Grouped Model Evaluations

After training, fitting, and testing of three random forest model variants, a weighted logistic regression, and a Naive Bayes classifier, the cumulative plot of ROC curves is displayed using the following command, drawing from a data frame variable housing model results named `result_table`.

```

#plot comparison figure
for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
             label="{}, AUC={:.3f}".format(result_table.loc[i]
['classifiers'], result_table.loc[i]['auc']))
plt.plot([0,1], [0,1], color='grey', linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title('ROC Curve Analysis')
plt.legend(loc=4)
plt.show()

```



From the cumulative ROC curve analysis, one can conclude that the random forests algorithms are superior in their ability to distinguish between positive and negative classes accurately in their predictions. Both the weighted logistic regression and the Naive Bayes classifiers struggle with precision in predicting the positive `OccDeathClassID`. More specifically, one can discern the following:

- **Random Forests Model Variants**

- **Manually Weighted Random Forest:** Has high accuracy and recall for both `OccDeathClassID` classifications in prediction. Suffers from precision in the positive predicted classification, and has a slight bias in its prediction towards the majority negative class in the form of type-1 errors. This is the most acceptable model given its propensity to retain an acceptable proportion of type-1 errors relative to true positives, all the while minimizing type-2 errors within acceptable thresholds.

- **Optimally Weighted Random Forest:** Has high accuracy and recall for both `OccDeathClassID` classifications in prediction. Does not suffer from poor precision in the positive predicted classification, and minimizes bias in its prediction towards the majority negative class in the form of few type-1 errors. Unfortunately, this model offsets the reduced type-1 errors with additional type-2 errors, which cannot be accepted given the context under study.

- **Balanced Random Forest:** Has high accuracy and recall for both `OccDeathClassID` classifications in prediction. Suffers from precision in the positive predicted classification, and has a considerable bias in its prediction towards the majority negative class in the form of type-1 errors. Loses out to the manually weighted random forests variant due to slightly lower accuracy, and marginally lower precision in identifying positive classifications correctly. Can be considered an overzealous model due to its likelihood of accepting a higher type-1 error threshold, but may be a candidate for production if the sole goal of the study is to minimize marine fatality occurrences.

- **Weighted Logistic Regression:** Suffers from poor precision in predicting the positive target class. Computationally expensive as many iterations need to be considered before a maximum likelihood estimate can be calculated. AUC value of 0.851 confirms

the model is more than capable of differentiating between positive and negative target classes, but makes more type-1 errors than acceptable for production.

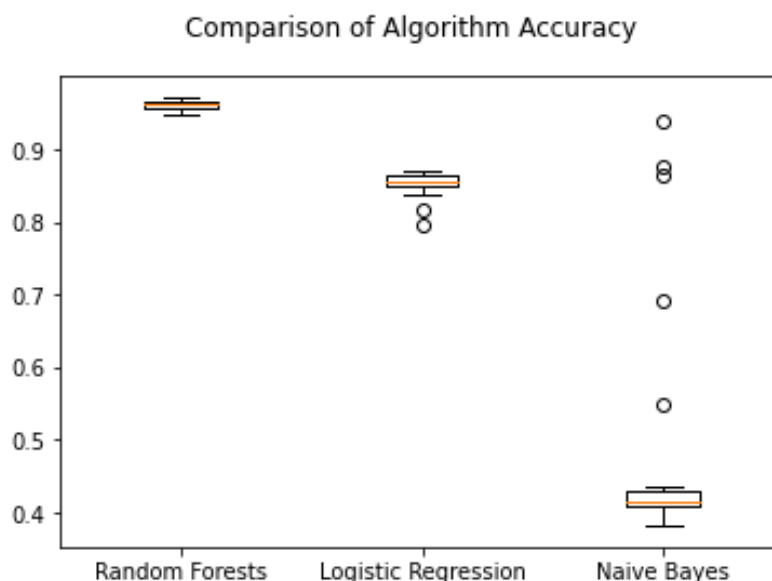
- **Naive Bayes Classifier:** Suffers from exceptionally poor precision in predicting the positive target class. Least computationally expensive model under consideration. AUC value of 0.611 confirms the model is marginally better than guessing the fatality outcome of a marine occurrence. Not recommended for additional consideration given the complexity of inter-variable interactions that the model is incapable of identifying.

The three model groupings' accuracy scores are visualized using the following command:

```
#store validation results in variables
results = []
results.append(n_scores)
results.append(n_log_scores)
results.append(n_nb_scores)

model_names = ['Random Forests', 'Logistic Regression', 'Naive Bayes']

#plot results to boxplot
fig = plt.figure()
fig.suptitle('Comparison of Algorithm Accuracy')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(model_names)
plt.show()
```



Final comparison of accuracy scores pertaining to each predictive model family.

Section 8: Conclusions

This study used several modelling techniques to accurately determine the relationship between key factors involved in Canadian marine occurrences and their contribution to marine accident fatalities. This study employed the use of three separate random forests algorithms, one weighted logistic regression, and one Naive Bayes classifier in order to determine the most appropriate model for predicting the positive target `OccDeathClassID` variable. Of the models under study, the Random Forests algorithm with a manually-specified class weighting hyperparameter was the most likely candidate in efficient prediction. Given its propensity to predict more accurately than other models, its feature importance set provides an understanding of the variables most conducive to a fatal marine occurrence in Canadian waters.

The top 5 influential attributes contributing to a marine accident fatality are ranked below in a descending fashion:

- 1. InjuriesIND (Injuries Indicator)** - Indicates whether the occurrence resulted in any injuries
- 2. AccIncTypeID (Accident/Incident Type)** - The type of accident or incident
- 3. Length_Meters (Length, in Meters)** - The length of the vessel involved in the marine incident, in Meters
- 4. OccurrenceTypeID (Occurrence Type)** - Indicates whether the occurrence was an accident or reportable incident
- 5. DamageIND (Damage Indicator)** - Indicates whether there was any damage to the vessel involved in the marine incident

There are two primary limitations regarding the approach offered in the study. Firstly, the validity of the data's completeness is brought under question given the high proportion of missing data in the features that were selected. Given additional time and consideration, a better method of imputation of NANS could be devised that does not rely on the mean of

encoded values. Secondly, it is possible that the accuracy of the data may be compromised to an undecipherable extent given the TSB's data reporting standards having the board report on itself. With no third party overseeing the recording processes and their accuracy's, some occurrence data may be underreported or reported incorrectly.

References

Government of Canada, T. S. B. of C. (1995, January 1). A SAFETY STUDY OF THE OPERATIONAL RELATIONSHIP BETWEEN SHIP MASTERS/ WATCHKEEPING OFFICERS AND MARINE PILOTS. Marine Investigation Report SM9501 - Transportation Safety Board of Canada. Retrieved October 23, 2022, from <https://www.tsb.gc.ca/eng/rapports-reports/marine/etudes-studies/SM9501/SM9501.html>

Government of Canada, T. S. B. of C. (2012, August 10). Marine investigation report M09Z0001. Marine Investigation Report M09Z0001 - Transportation Safety Board of Canada. Retrieved October 23, 2022, from <https://www.tsb.gc.ca/eng/rapports-reports/marine/etudes-studies/m09z0001/m09z0001.html>

Government of Canada, T. S. B. of C. (2019, May 6). Marine Transportation Safety Investigations and reports. Transportation Safety Board of Canada. Retrieved October 23, 2022, from <https://www.tsb.gc.ca/eng/rapports-reports/marine/index.html>

International Maritime Organization. (2014, November 18). CASUALTY-RELATED MATTERS* REPORTS ON MARINE CASUALTIES AND INCIDENTS Revised harmonized reporting procedures – Reports required under SOLAS regulations I/21 and XI-1/6, and MARPOL, articles 8 and 12. International Maritime Organization. Retrieved October 23, 2022, from <https://wwwcdn.imo.org/localresources/en/OurWork/MSAS/Documents/MSC-MEPC3/MSC-MEPC.3-Circ.4%20Rev%201%20%20Revised%20harmonized%20reporting%20procedures%20-%20Reports%20required%20under%20SOLAS%20regulations%20I21.pdf>

Lemaitre, G., & Aridas, C. (2022, May 22). *imbalanced-learn 0.9.1*. PyPi. Retrieved from <https://pypi.org/project/imbalanced-learn/>

Transportation Safety Board of Canada. (1995). Figure 1. Transportation Safety Board of Canada. Government of Canada. Retrieved October 23, 2022, from <https://www.tsb.gc.ca/eng/rapports-reports/marine/etudes-studies/SM9501/images/ems9501a.gif>.

Transportation Safety Board of Canada. (2010). Figure 4. Safety Issues Investigation into Fishing Safety in Canada. Government of Canada. Retrieved October 23, 2022, from <https://www.tsb.gc.ca/eng/rapports-reports/marine/etudes-studies/M09Z0001/images/m09z0001-figure-04.png>.

Transportation Safety Board of Canada. (2019, June 12). Marine occurrence data from January 1995 to present. Open Government Portal. Retrieved October 23, 2022, from <https://open.canada.ca/data/en/dataset/ad8d1b73-df09-4521-9bdb-61c529328218>

Transportation Safety Board of Canada. (2019, June 12). Marine occurrence data from January 1995 to present. Open Government Portal. Retrieved October 23, 2022, from <https://open.canada.ca/data/en/dataset/ad8d1b73-df09-4521-9bdb-61c529328218>

Wang, H., Liu, Z., Wang, X., Graham, T., & Wang, J. (2021). An analysis of factors affecting the severity of marine accidents. *Reliability Engineering & System Safety*, 210, 107513. doi:10.1016/j.ress.2021.107513

Github Repository

All relevant project files can be found at <https://github.com/rfinatan/CIND-820-Big-Data-Analytics-Project>.